VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Animace detekčních a korekčních metod v přenosu dat
Animation of detection and error correction methods in data transmission

2011 Roman Kovalovský

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání diplomové práce

Student: **Bc. Roman Kovalovský**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2601T013 Telekomunikační technika

Téma: Animace detekčních a korekčních metod v přenosu dat
Animation of detection and error correction methods in data transmission

Zásady pro vypracování:

Při přenosu dat je velmi důležité pochopit základní principy detekce a opravy dat. Cílem této práce je vytvořit animace, popisující některé metody.
1. Úvod do problematiky.
2. Popis různých metod.
3. Vytvoření vybraných animací.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplonové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Pavel Nevlud**

Datum zadání: 19.11.2010
Datum odevzdání: 06.05.2011

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA *

prof. RNDr. Vladimír Vašinek, CSc.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení

„Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.“

V Ostravě ……. Podpis studenta …….

Poděkování

Rád bych poděkoval svému vedoucímu diplomové práce Ing. Pavlu Nevludovi za metodickou, pedagogickou a odbornou pomoc a za další cenné rady při zpracování mé diplomové práce.

V Ostravě ……. Podpis studenta …….

Abstrakt

Práce se zabývá bezpečnostními kódy, které zabezpečují přenos dat proti chybám. Nejprve je vysvětlen rozdíl mezi korekčními a detekčními metodami. Následně jsou podrobněji popsány metody Cyklického redundantního součtu, Hammingůva kódu, Reed-Müllerova kódu a Konvolučních kódů. Pro všechny tyto metody jsou popsány základní vlastnosti, vytvoření kódového slova a příjem kódového slova. K těmto metodám jsou vytvořeny animace, ve kterých je ukázáno a vysvětleno, jak jednotlivé metody zakódují informační posloupnost a následně ji dekódují.

Klíčová slova

bezpečnostní kódy, CRC, Hammingův kód, Reed-Müllerův kód, konvoluční kód, detekční kód, korekční kód, přenos dat, animace

Abstract

My thesis deals with Forward Error Correction which secure data transmission against errors. First of all the diffrence between correction a detection methods is explaned. After that methods of Cyclic Redundancy Check, Hamming code, Reed-Muller code and convolution codes are described more in detail. Basic properties, creation of the code word and reception the code word are described for all the above mentioned methods. The animation I have made, present and explain how each single methods encodes inforamtion order and decodes it.

Keywords

safety code, CRC, Hamming code, Reed-Muller code, convolutional code, detection code, forvard error corrections, data transmission, animation

Seznam použitých symbolů a zkratek

ARQ	Potvrzovací metoda (Automatic request for retransmission)
B(x)	Kódové slovo
B´(x)	Přijaté kódové slovo
CRC	Cyklický redundantní součet (Cyclic redundancy check)
E	Jednotková matice
G	Generující matice
G(x)	Generující (vytvářecí) polynom
H	Kontrolní matice
H´	Ekvivalentní matice
M(x)	Blok informačních symbolů
Q(x)	Výsledný polynom po dělení
R(x)	Zbytek po dělení
S(x)	Kontrolní syndrom
k	počet informačních bitů
k_0	Počet vstupů
m	počet zabezpečujících bitů
n	počet všech bitů kódového slova
n_0	Počet výstupů
s	Kontrolní syndrom
s^T	Transponovaný vektor kontrolního syndromu
u	Blok informačních symbolů
v	Odeslané kódové slovo
w	Přijaté slovo
x^r	Řád polynomu

Obsah

1. Úvod

V dnešní době se stal přenos informací na vzdálená místa součástí našeho života. Rychlý životní styl vyžaduje přenesení objemu informací v co nejkratší době. K tomu využíváme kompresní metody a snažíme se zbavit veškeré redundance. Na druhé straně nám vzniká problém s bezchybným přenosem dat. Při přenosu dochází k různým rušivým vlivům, ať už pomocí metalického vedení, optického kabelu nebo radiovými vlnami. Tyto nežádoucí vlivy nám znehodnocují užitečný signál a dochází k zavádění chyby. Odstraněním tohoto problému se zabývají bezpečnostní kódy, které pro změnu k užitečné informaci přidávají jistou nadbytečnost, na základě které na přijímací straně dokážou detekovat nebo dokonce opravovat vzniklé chyby. Detekci a korekci chyb řeší teorie kódování, která má počátky už ve čtyřicátých letech. Dnes je teorie kódování velmi rozšířená a zabývá se odstraňováním chyb, zrychlením přenosu dat, utajením informace a samotnými konstrukcemi kódů.

Bezpečnostní kódy jsou děleny do několika skupin dle vlastností jednotlivých kódů. Obecněji jsou rozděleny do dvou skupin, kterými jsou detekční metody a korekční metody. Rozdělují se na základě způsobu opravy vzniklé chyby. Podrobněji jsou popsány v následující kapitole.

V další části práce je uveden detailnější popis konkrétních metod, jejich vlastností, vytvoření kódového slova a objevení nebo i opravení chyby vzniklé během přenosu dat. Práce se soustředí na Cyklický redundantní součet, Hammingovy kódy, Reed-Müllerovi kódy a poslední skupinou jsou konvoluční kódy.

Závěrečná kapitola je věnována výukovým animacím, ve kterých je prakticky ukázána a vysvětlena funkce a princip zmíněných metod. Vytvořené animace budou sloužit jako doplňkový výukový materiál pro studenty.

Příloha je věnovaná příkladům výpočtu kódových slov a jejich následné detekci nebo korekci chyb dle jednotlivých metod. Pro výpočet je zvolena stejná vstupní informační posloupnost, z důvodu srovnání rozdílů vypočítaných kódových slov.

2. Bezpečnostní kódy

Díky rušivým vlivům při přenosu číslicového signálu vznikají chyby. V jejich důsledku může příjemce přijmout jiné znaky, než jaké mu odesílatel původně vyslal. Pro příjemce má však význam pouze bezchybně přijatá zpráva a tak je třeba přenos nějak zabezpečit. Těmto chybám můžeme čelit požitím bezpečnostních kódů, které vlivem uměle zvýšené redundance dokážou na přijímací straně chyby detekovat nebo úplně odstranit [6]. Jinými slovy k užitečné informaci přidají takovou informaci, která na přijímací straně dokáže zprávu s chybou rozpoznat nebo zrekonstruovat na původní zprávu bez chyby.

2.1 Detekční kódy

Kladou si za cíl pouze detekovat vzniklou chybu. Způsob opravy chyb je založen na existenci zpětné vazby a na tom, že příjemce si vynutí opakované vyslání těch dat, které se při dřívějším přenosu poškodily. Postup je obvykle takový, že odesilatel vysílá data po blocích určité velikosti a příjemce vrací odesilateli informaci o tom, zda konkrétní blok došel v pořádku či nikoli. K tomuto postupu se dá využít mechanizmus pro opravu chyb na základě automatické žádosti o opakování tzv. ARQ metody. Tyto detekční kódy se mohou využít tam, kde je možnost zavedení zpětné vazby a lze ovlivnit přenosovou rychlost mezi odesílatelem a příjemcem, protože chybná data se musí posílat opakovaně.

2.2 Korekční kódy

Korekční kódy navíc oproti detekčním umějí chybu nejen detekovat, ale také ji opravit. Tato oprava spočívá v tom, že odesílaným datům je připojena taková informace, která při výskytu chyby umožní příjemci data zrekonstruovat na původní odesílaná data. Výhodou korekčních kódů nebo také samoopravných kódů je, jak uvádí některé knihy, že se o tuto rekonstrukci stará pouze přijímací strana, tudíž není nutné data odesílat znovu. Mezi nevýhody

pak lze zařadit fakt, že korekční kódy dokážou opravit chyby pouze určitého rozsahu. S chybami většího rozsahu si nedokážou poradit. Navíc zde platí vcelku zřejmá závislost. Čím větší chyby dokáže korekční kód opravit, tím větší je objem dat, které je nutné „přibalit" k původním datům pro potřeby jejich případné rekonstrukce. Objem těchto „přibalených" dat, ale v praxi vychází neúnosně velký. Proto se používají korekční kódy tam, kde potřebujeme zvýšit spolehlivost doručených dat a zároveň není přijatelné nebo příznivé opakovat přenos poškozených paketů, protože existuje jen jeden směr přenosu (simplex) nebo by opakovaný přenos neúměrně snížil přenosovou rychlost, tam kde si to nemůžeme dovolit.

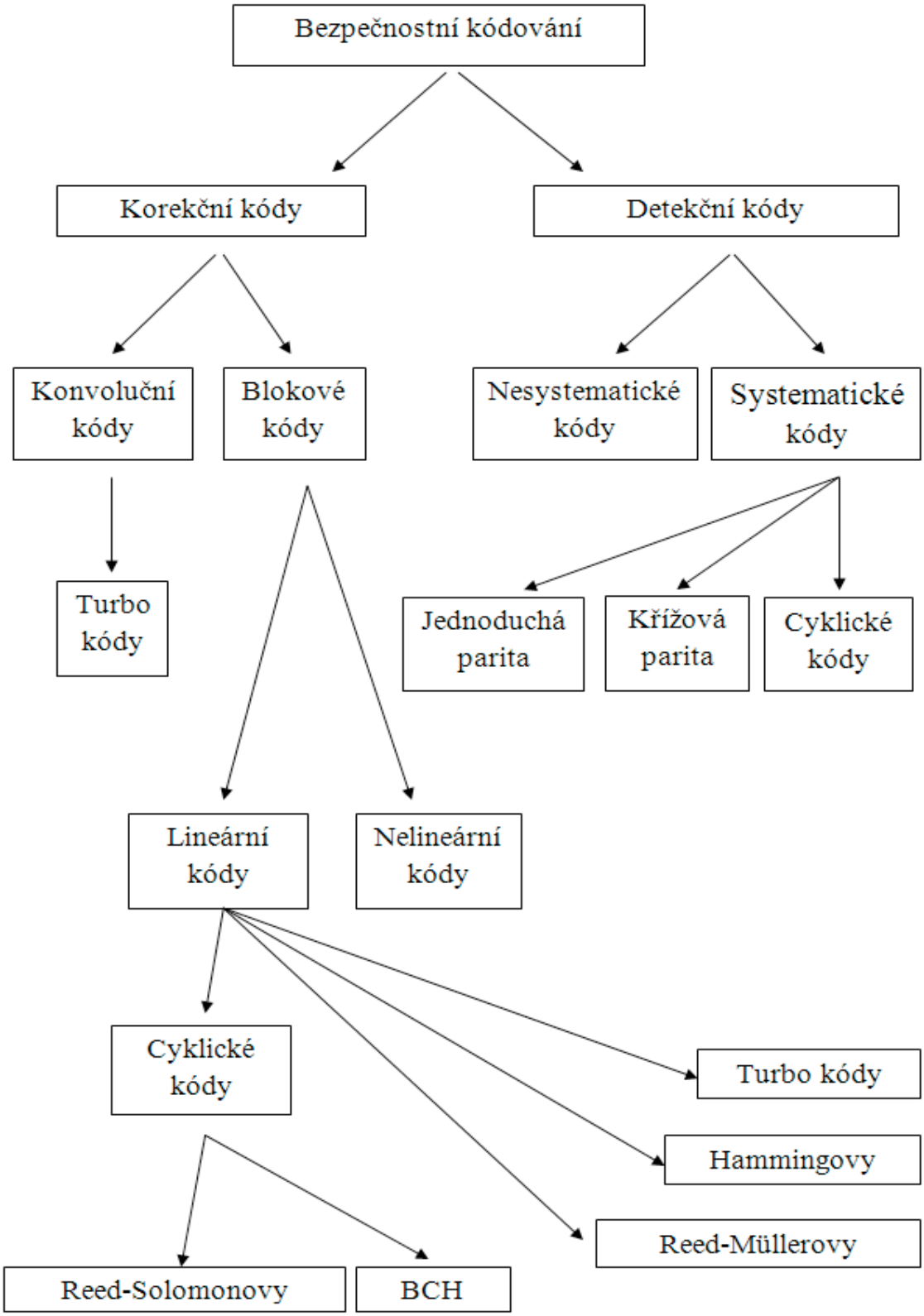
2.3 Kódování

Definice kódování:
Je zobrazení K:A→B*, jinak je to předpis, který každému prvku konečné množiny A přiřadí právě jedno slovo z množiny B*.[5]

kde
– A je zdrojová abeceda
– B* jsou slova sestavená z kódové abecedy B

2.4 Rozdělení některých bezpečnostních kódů

V grafu 2.1 jsou uvedeny základní rozdělení některých bezpečnostních kódů.



Graf 2.1: Rozdělení bezpečnostních kódů

Hammingovy kódy - Jedná se o blokové korekční kódy, dokážou opravit jednu chybu a jejich minimální Hammingova vzdálenost je $d_{min} = 3$. Jejich hlavní výhodou je, že dokážou opravit jednoduché chyby s nejmenší myslitelnou redundancí. Takovéto kódy se nazývají perfektní kódy.

Reed-Millerův kód – Reed-Müllerův kód je lineární kód definovaný nad Galoisovým tělesem GF(2) [1], které mohou být dekódovány jednoduchou technikou rozhodování. Z těchto důvodů jsou Reed-Müllerovy kódy důležité i přesto, že někdy nesplňují nejmenší kódovou vzdálenost.

Konvoluční kód - Konvoluční kodéry lze popisovat jen jako zdroje zpráv s pamětí. Konvoluční kódy jsou předpisem pro kódový systém, který generuje kódová slova na základě obsahu rámce několika vstupních slov. To jakým způsobem bude zakódována určitá část informační posloupnosti tedy záleží nejenom na aktuální vstupní propusti, ale také na předchozích vstupních informačních slovech.

Bose-Chaudhuriho-Hocquenghemovy kódy - (BCH kódy) – Cyklické, blokové, binární kódy, které zvládají opravu vícenásobných chyb. Mezi jejich přednosti patří velká volitelnost parametrů, dobrý vztah mezi počtem informačních znaků a počtem opravovaných chyb a detailně vypracované dekódovací metody. Jejich název je zkracován na BCH kódy.

Reed-Solomonovy kódy – Vychází z BCH kódů a jejich předností je, že mají nejlepší opravné schopnosti pro kód dané délky. Ve srovnání s BCH kódy mají však složitější způsob dekódování.

Turbo kódy – Tyto kódy vznikají paralelním zřetězením dvou nebo více kódů za použití prokládání. Mohou být při tom využiti, jak blokové, tak i konvoluční kodéry. Před uvedením turbo kódů byly nároky na vysokou účinnost potlačení chyb v informačním kanálu řešené například konvolučním kódem velkou hodnotou omezující délky kódu. V roce 1993 byly poprvé popsány tzv. turbo-kódy. Dnes patří Turbo-kódy k nejvýznamnějším objevům v teorii kódovaní.

3. Cyklický redundantní součet

Cyklický redundantní součet, který můžeme znát pod zkratkou CRC se používá k detekci chyb během přenosu dat. Jedná se pouze o detekční kód. Chybu umí pouze detekovat, nikoliv ji opravit. CRC není cyklický kód, jako by se mohlo na první pohled zdát podle slova cyklický, ale pouze z cyklického kódu vychází. Jedná se systematický kód, proto v odeslaném kódovém slovu vidím přímo blok informačních bitů. U CRC se využívá výhradně sériového zpracování. V praxi jde o velmi rozšířený a požívaný kód a to z důvodu jednoduchosti a dobrých matematických vlastností. CRC je velmi úspěšné při odhalení chyby včetně shlukových chyb a to s úspěšností téměř 100% viz tabulka 3.2. Princip je založen na kontrolním součtu, který je posílám společně s informačním blokem M(x) dle [2]. Po dokončení přenosu dat je tento součet znovu spočítán a je-li odlišný od přeneseného, pak při přenosu došlo k chybě. Je-li však stejný, přenos proběhl bez chyby. Abychom získali tento kontrolní součet, musíme dělit polynom polynomem. Při této matematické operaci platí základní pravidla uvedená v tabulce 3.1.

$x^i + x^i = 0$
$x^i + x^i + x^i = x^i$
$x^i \cdot x^y = x^{i+y}$
$x^i \div x^y = x^{i-y}$

Tabulka 3.1: Základní matematické operace pro polynomy

CRC je tedy založen na dělení v konečném Galoisově tělese $GF(2^n)$. [1]. Což je těleso polynomů nad celými čísly modulo 2, jinak řečeno je to množina polynomů, jejichž koeficienty mohou nabývat pouze hodnot 1 a 0. Tyto polynomy můžeme sčítat, odčítat, násobit a dělit jako obyčejné polynomy, ale nad jejich výsledky provádíme operaci modulo 2, tedy zbytek po dělení dvěma. $(x^2 + x) + (x + 1) = x^2 + 2x + 1 = x^2 + 1$ Zde můžeme názorně vidět, jak se ze dvojky stane nula. Nad koeficienty se provádí operace modulo 2. Stejné to je i při jiných matematických operacích např. násobení.

Schopnost detekce chyb je závislá na generujícím polynomu. Při správné volbě hodnoty mají delší klíče lepší schopnost detekce chyb. Za písmeny CRC můžeme vidět číslo. Např. CRC12 má generující polynom stupně 12, což značí nejvyšší koeficient x^{12}. Při některých způsobech výpočtu se za vstupní data přidává stejný počet nul v závislosti na řádu generujícího polynomu, resp. dojde k vynásobení informačního bloku řádem polynomu. Je to z toho důvodu, že vypočtené CRC ze vstupních dat a uloženého CRC je pak nulové.

Druh CRC	CRC-7	CRC-12	CRC-16	CRC-32
Úspěšnost odhalení chyby	99,2188 %	99,9756 %	99,9985 %	99,9999 %

Tabulka 3.2: Úspěšnost odhalení chyby

3.1 Vytváření kódových slov

Zde můžeme vidět základní vztahy pro vyslání kódového slova dle vzorců (3.1, 3.2 a 3.3):

$$B(x) = x^r M(x) \oplus R(x) \tag{3.1}$$

$$\frac{B(x)}{G(x)} = Q(x) \tag{3.2}$$

$$\frac{x^r M(x)}{G(x)} = Q(x) + \frac{R(x)}{G(x)} \tag{3.3}$$

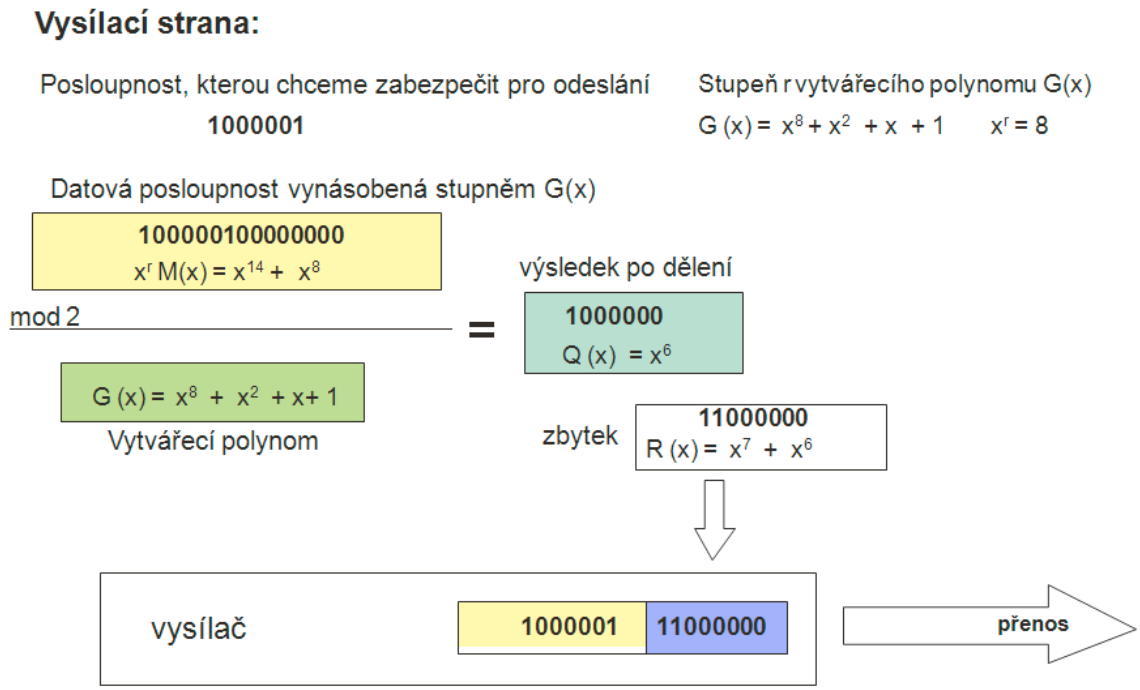
Při vytváření kódového slova vycházíme z vlastnosti, že se jedná o systematický kód. To znamená, že k bloku informačních bitů M(x) viz obrázek 3.1 musíme vypočítat část zabezpečovací, což je v tomto případě část CRC. Klíčem k této hodnotě je generující polynom G(x). Tímto generujícím polynomem vydělíme blok informačních bitů. Výsledek dělení těchto dvou polynomů pro vytvoření kontrolního součtu není důležitý, k vytvoření kontrolního součinu využijeme pouze zbytek po dělení. Právě tento zbytek po dělení je naše hledané CRC, kterým zabezpečíme přenos dat. Před samotným dělením nesmíme zapomenout rozšířit blok informačních symbolů o řád generujícího polynomu, který je roven nejvyššímu stupni mocniny generujícího polynomu. Tímto krokem si usnadníme detekci na přijímací straně a v případě

bezchybného přenosu bude výsledný zbytek po dělení roven nule. Další nezbytnou podmínkou pro zachování správné funkce je, že generující polynom musí znát i strana, která zajišťuje příjem dat. Bez stejného generujícího polynomu by nebyla schopna korektně vydělit přijatá data a následně detekovat chybu.

Informační blok	CRC

Blokové schéma 3.1: Rozmístění skupin bitů při přenosu

K tomu, abychom mohli lépe a přehledněji dělit polynomy je výhodné si bitové posloupnosti převést právě na polynomy. Ovšem na výsledek to nemá žádný vliv, protože polynomy a bitové posloupnosti jsou ekvivalentní. Právě jednoduchá implementace operací nad bitovými posloupnostmi je jedním z hlavních důvodů širokého rozšíření CRC algoritmů.



Obrázek 3.1: Blokové schéma vzniku kódového slova

3.2 Oprava kódových slov

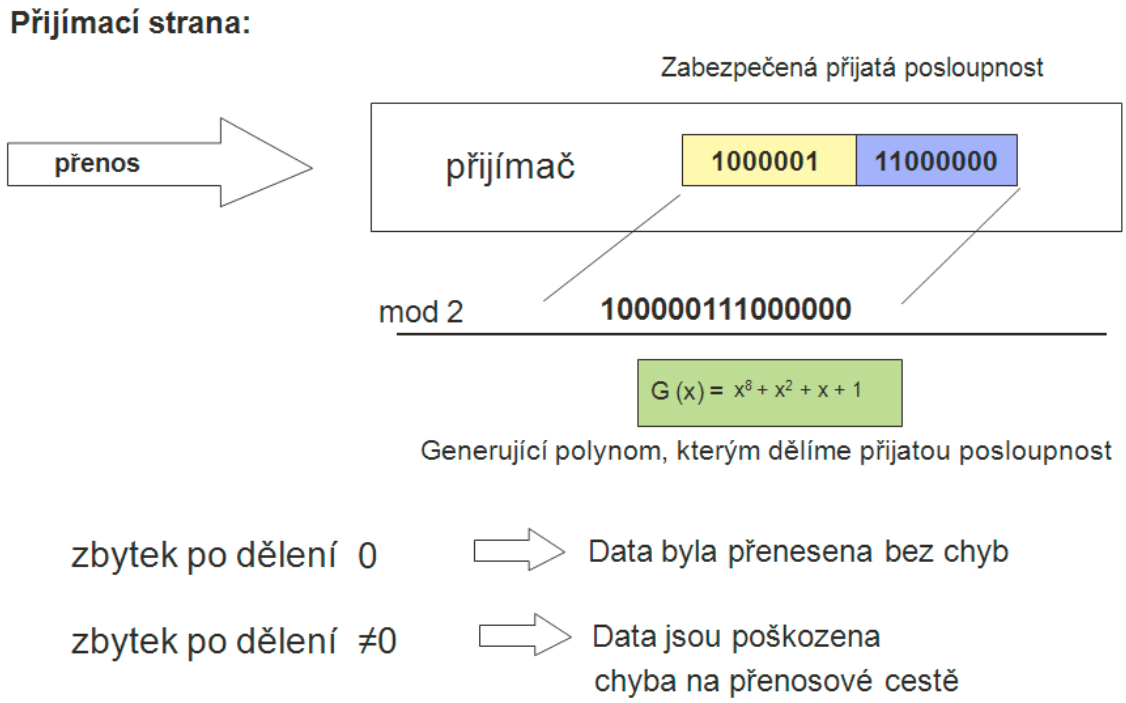
Základní vzorce pro příjem kódového slova.

$$B'(x) = B(x) \oplus E(x) \tag{3.4}$$

$$\frac{B'(x)}{G(x)} = Q'(x) + \frac{S(x)}{G(x)} \tag{3.5}$$

Na straně, která se stará o příjem dat, se tato přijatá posloupnost znovu překontroluje. Princip dékódování je naznačen na obrázku 3.2. Pomocí těchto vzorců 3.4 a 3.5 vypočítáme kontrolní syndrom, pokud je:

- S(x)=0 → při přenosu nedošlo k chybě nebo došlo k nedetekovatelné chybě
- S(x)≠ 0 → při přenosu došlo k chybě



Obrázek 3.2: Blokové schéma dekódování CRC

4. Hammingův Kód

Hammingův kód je binární lineární kód, který má uplatnění v telekomunikacích. Jde o velmi významnou třídu kódů. Vlastnosti Hammingova kódu jsou dány minimální kódovou vzdáleností $d_{min} = 3$. Tato vzdálenost nám zaručuje opravu jedné chyby a detekci dvou chyb dle vzorce (4.1, 4.2 a 4.3).

Vzorec pro opravu chyby $$d \geq 2t + 1 \quad (4.1)$$

nebo $$t \langle \frac{d}{2} \quad (4.2)$$

kde *d* je vzdálenost kódových slov, *t* je počet chyb.

Vzorec pro detekci chyby $$d = t + 1 \quad (4.3)$$

kde *d* je vzdálenost kódových slov, *t* je počet chyb

Tyto kódy jsou blokové, což znamená, že všechna kódová slova mají stejnou délku *n*. V kódovém slovu vidíme přímo část informačních bitů a to znamená, že tyto kódy jsou systematické. Hammingovy kódy se neobyčejně snadno dekódují a jsou perfektní, tj. mají nejmenší možnou myslitelnou redundanci [1].

Hammingův binární kód má více konfigrací viz tabulka 4.1, pokud budeme zvětšovat počet zabezpečujících bitů *m* (m = 2, 3, 4), potom dostáváme binární (3, 1)-kód, (7, 4)-kód, (15, 11)-kód atd.

m	2	3	4	5	6
n	3	7	15	31	63
k	1	4	11	26	57

Tabulka 4.1: Konfigurace Hammingových kódů

kde

m – počet zabezpečujících prvků

n – počet všech prvků

k – počet informačních prvků

Vzorec pro výpočet informačních prvků v kódovém slovu

$$k = n\text{-}m \tag{4.4}$$

Vzorec pro výpočet všech prvků v kódovém slovu

$$n = 2^m\text{-}1 \tag{4.5}$$

Informační poměr je dán poměrem počtu informačních znaků k počtu všech znaků kódového slovu a roste rychle k jedné [1].

Vzorec pro výpočet informačního poměru

$$R = \frac{k}{n} = \frac{57}{63} = 0{,}9 \quad [\text{-};\text{-},\text{-}] \tag{4.6}$$

Sloupce kontrolní matice musí splňovat dvě podmínky:

- Sloupce musí být nenulové (podmínka pro detekci jedné chyby) a musí být nenulový součet dvou sloupců (podmínka dvojnásobné chyby).
- Sloupce kontrolní matice musí být různé a žádný z nich se nesmí opakovat.

Kontrolní matice Hammingova kódu

$$H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

K sestavení generující matice musíme kontrolní matici H upravit tak, abychom dostali v posledních třech sloupcích jednotkovou matici. Toho docílíme tím, že zaměníme 1. sloupec za 7.sloupec, 2.sloupec za 6.sloupec a 4.sloupec za 5.sloupec. Dostaneme matici H´, která je maticí ekvivalentního Hammingova kódu.

$$H' = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Z matice ekvivalentního Hammingova kódu H´ = [B^T| E] můžeme vypočítat generující matici G´ = [E | B]

$$G' = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Generující matici původního kódu získáme opětovným přehozením sloupců 1. sloupec za 7. sloupec, 2. sloupec za 6. sloupec a 4. sloupec za 5. sloupec.

$$G = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Matice ve tvaru G´ je pro nás výhodnější z hlediska kódování informačních znaků, protože Hammingův kód je systematický. Naproti tomu při dekódování je výhodnější tvar matice G nebo matice H, protože generující matici nepotřebujeme znát. To je dáno tím, že kódová slova generující matice G lze získat jako řádky této matice a jejich lineární kombinace. Celkem těmito lineárními kombinacemi dostane 16 kódových slov (viz. tabulka 4.2). Mezi kódové slovo se řadí i nulové slovo. Další již nenulová slova vzniknou jako součty řádků generující matice G.

sčítané řádky	kódové slovo	sčítané řádky	kódové slovo
1,1	0	1,2	1000011
1,2,4	1111	4	1001100
2,3,4	10110	1,3,4	1010101
1,3	11001	2,3	1011010
1,4	100101	2,4	1100110
2	101010	1	1101001
1,2,3	110011	3	1110000
3,4	111100	1,2,3,4	1111111

Tabulka 4.2: Kódová slova Hammingova kódu (7, 4)

4.1 Vytváření kódových slov

Při vytváření kódových slov využijeme toho, že se jedná o blokový a systematický kód. K informačním bitům dopočítáme zabezpečovací bity. Jednotlivé zabezpečovací bity jsou vytvářeny relacemi, které můžeme získat ze soustavy vyjádřené kontrolní maticí H. Násobením kontrolní matice H s transponovaným vektorem kódového slova a po další úpravě matice H dostaneme vyjádření pro zabezpečovací bity [1].

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \end{bmatrix} = 0$$

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 & \oplus \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & \uparrow \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & \downarrow \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & \oplus \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & \oplus \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & \uparrow \end{bmatrix} \rightarrow$$

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Schéma 4.1:Schéma úpravy kontrolní matice H na kanonický tvar

Kontrolní matici H musíme upravit na tzv. kanonický tvar. Postupné úpravy kontrolní matice H jsou naznačeny ve schématu 4.1. Výsledkem bude matice, kde zabezpečovací bity v_5 v_6 a v_7 pak získáme ze zdrojových bitů v_1 až v_4 a bude mít následující tvar:

$$[v_5 v_6 v_7] = [v_1 v_2 v_3 v_4] \cdot \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Díky těmto rovnicím lze ověřit, zda kódové slovo patří do Hammingova kódu (7,4).

Rovnice pro výpočet zabezpečovacích bitů:

$$v_5 = v_2 \oplus v_3 \oplus v_4 \tag{4.7}$$

$$v_6 = v_1 \oplus v_3 \oplus v_4 \tag{4.8}$$

$$v_7 = v_1 \oplus v_2 \oplus v_4 \tag{4.9}$$

Při použití tohoto postupu vypočítáme všechny kódová slova Hammingova kódu (7, 4), které si můžeme ověřit v tabulce 4.2.

V praxi se používá také Hammingův kód (15, 11). Platí pro něj stejná pravidla, jako pro Hammingův kód (7, 4) a s jeho kontrolní maticí uvedenou pod tímto odstavcem pracujeme stejně. Zvolíme-li $m = 4$, dostáváme (15,11)-kód s kontrolní maticí.

Matice Hammingova kódu (15, 11)

$$H = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

4.2 Příjem kódových slov

Binární Hammingovy kódy se velmi snadno dekódují pomocí tzv. syndromu [3]. Tento syndrom dostaneme vynásobením kontrolní matice H s transponovaným vektorem přijatého slova w^T dle vzorce (4.10). Pokud je hodnota součinu nula, potom přenos proběhl bez chyby. Ovšem pokud je hodnota syndromu nenulová, pak při přenosu nastala chyba. Jeden chybně přenesený bit v přijatém slovu způsob, že se přijaté slovo bude mít od odeslaného slova vzdálenost 1. Ovšem tento jeden chybný bit umí Hammningův kód opravit.

Vzorec pro výpočet syndromu

$$s^T = H\, w^T \tag{4.10}$$

Předpokládejme, že při přenosu došlo k jednonásobné chybě a přijali jsme slovo w = [0110101].

$$s^T = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \\ w_7 \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} \qquad s \begin{cases} \neq 0 \rightarrow došlo\, k\, chybě \\ = 0 \rightarrow nedošlo\, k\, chybě \end{cases}$$

Vynásobením kontrolní matice H a přijatého slova w, které musíme převést na transponovaný tvar, tzn. z řádků se stanou sloupce a aplikováním vzorce (4.10) dostaneme hodnotu syndromu. Tento syndrom musí být roven sloupci kontrolní matice, jehož pořadí v matici je rovné pořadí chybného bitu v přijatém slově. Porovnáváním syndromu se sloupci kontrolní matice vyplívá, že syndrom je *s* shodný s třetím sloupcem kontrolní matice.

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

Hodnota syndromu detekuje chybu na třetím místě. Na této pozici provedeme opravu za správný znak a tím opravíme přijaté slovo na správné kódové slovo v = [0100101].
Pokud by bylo přijaté slovo w = [0111101], znamená to, že při přenosu došlo ke dvojnásobné chybě. Hodnota syndromu by byla [111]. Jelikož je tento syndrom nenulový, znamená to, že při přenosu nastala chyba. Pokud bychom chtěli tuto chybu opravit, došlo by k této opravě na nesprávném místě. Kód není schopen opravovat dvojnásobné a vícenásobné chyby
viz vzorec 4.1.

4.3 Rozšířený Hammingův kód

Rozšířený Hammingův kód má minimální vzdálenost $d = 4$. Z toho plyne, že umí opravit jednu chybu viz vzorec 4.1 a detekovat až trojnásobné chyby viz vzorec 4.3. Kontrolní matici rozšířeného Hammingova kódu dostaneme tak, že k původní matice Hammingova kódu přidáme každému řádku bit sudé parity a spodní řádek kontrolní matice doplníme řádkem jedniček [9]. Rozšířeného Hammingova kódu se hojně využívá v praxi. Kontrolní matice musí splňovat určité podmínky, sloupce kontrolní matice musí být nenulové, tím je zajištěna detekce jedné chyby, součet dvou sloupců nesmí být nulový vektor, tím je zajištěna detekce

dvojnásobné chyby a poslední podmínka je, že součet tří sloupců nesmí být nulový vektor, tím se zajistí detekce třínásobné chyby.

Kontrolní matice pro rozšířený Hammingův kód (8,4) má tedy tvar:

$$H_4 = \begin{bmatrix} & & & & & & & 0 \\ & & & H_3 & & & & 0 \\ & & & & & & & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

H_3 je původní kontrolní matice Hammingova kódu (7,4). Při vytváření kódových slov rozšířeného Hammingova kódu (8,4) budeme postupovat následovně. Kontrolní matici Hammingova kódu (7,4) rozšíříme na kanonický tvar. Postup je stejný jako v předcházejícím případě, k matici v tomto tvaru přičteme první tři řádky ke čtvrtému řádku. Touto úpravou dostaneme kanonický tvar kontrolní matice rozšířeného Hammingova kódu (8,4).

$$\begin{bmatrix} & & & & & & & 0 \\ & & & H_3 & & & & 0 \\ & & & & & & & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \rightarrow$$

$$\rightarrow \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Kontrolní bity v_5- v_8, pak vypočítáme z informačních bitů v_1-v_4 dle rovnic:

$$v_5 = v_2 \oplus v_3 \oplus v_4 \tag{4.11}$$

$$v_6 = v_1 \oplus v_3 \oplus v_4 \tag{4.12}$$

$$v_7 = v_1 \oplus v_2 \oplus v_4 \tag{4.13}$$

$$v_8 = v_1 \oplus v_2 \oplus v_3 \tag{4.14}$$

Všechna kódová slova rozšířeného Hammingova kódu (8,4), vypočítaná podle těchto rovnic jsou uvedena v tabulce 4.3.

00000000	01001011	10000111	11001100
00011110	01010101	10011001	11010010
00101101	01100110	10101010	11100001
00110011	01111000	10110100	11111111

Tabulka 4.3: Kódová slova rozšířeného Hammingova kódu (8,4)

Rozšířený Hammingův kód (16,11) se vytváří stejným způsobem jako kód (8,4). Rozšířený Hammingův kód (jakéhokoliv rozměru) není perfektním kódem.

5. Reed-Müllerův kód

Reedovy-Müllerovy kódy jsou nejstarší známou třídou kódů, opravují vícenásobné chyby. Byly objeveny v roce 1954. Představují velkou třídu binárních lineárních kódů. Jejich způsob dekódování je relativně snadný na základě techniky majoritního rozhodování. Tento kód je blokový, všechna kódová slova mají stejnou délku. Pro každé celé m a pro každé celé r má tento kód délku 2^m a nejmenší kódovou vzdálenost $d_{min} = 2^{m-r}$. Tento kód je nesymetrický, tudíž v kódovém slovu nevidíme přímo část informačních bitů. Reedův-Müllerův kód je definován pomocí konstrukce generující matice. Definice Reedových-Müllerových kódů je založena na boolovských funkcích více [3]. Reedovy-Müllerovy kódy byly použity v kosmickém korábu Mariner 9 při posílání fotografií z planety Marz. Základní výpočty a parametry kódu, ze kterých dostáváme různé konfigurace kódu viz. tabulka 5.1

délka kódového slova $$n = 2^m \tag{5.1}$$

řád $$r < m \tag{5.2}$$

počet informačních bitů $$k = 1 + \binom{m}{1} + \cdots + \binom{m}{r} \tag{5.3}$$

počet chyb, které můžeme opravit $$t = 2^{m-r-1} - 1 \tag{5.4}$$

m	4	4	3	3
n	16	16	8	8
r	1	2	1	2
k	5	11	4	7

Tabulka 5.1: Konfigurace Reedova-Müllerova kódu

kde

n…. délka kódového slova

r….řád

k….počet informačních bitů

t….počet chyb, které můžeme opravit

5.1 Vytváření kódových slov

Generující matice Reedova-Müllerova kódu se skládá z dílčích submatic. Nechť máme kód řádu *r* a délky 2^m, pak tato matice vzniká z *r* + 1 submatic.

$$G = \begin{bmatrix} G_0 \\ G_1 \\ \cdots \\ \cdots \\ G_r \end{bmatrix}$$

Submatice G_0 je vektor obsahují pouze jedničky o délce 2^m. Submatice G_1 má rozměr *m* x 2^m a obsahuje všechny binární *m*-tice v každém sloupci pouze jednou. Do sloupec nejvíce vlevo obsahuje pouze nuly a sloupec nejvíce vpravo zase samé jedničky. Submatice G_2 obsahuje součiny všech řádků submatice G_1 právě jednou. Výsledná generující matice je pak spojení jednotlivých submatic dohromady.

Součin *I* řádků, matice G_I má pak rozměr $\binom{m}{I}$ (5.5)

Je tedy možné použít $\binom{m}{I}$ způsobů, jak vytvořit součin *I* řádků, matice G_I má pak rozměr $\binom{m}{I}$ řádků a 2^m sloupců.

Kódové slovo pak určíme operací $B(x) = M(x) \cdot G$ (5.6)

Ukažme si příklad vytvoření generující pro parametry $m = 4$ a $r = 3$.

Submatice G_0 je vektor délky $n=2^m$, tedy $2^4 = 16$ bitů, které tvoří samé jedničky

$$G_0 = [1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1]$$

Submatice G_1 má rozměr $m \times 2^m$ a do sloupce, který je nejvíce vlevo umisťujeme nuly, sloupec nejvíce vpravo obsahuje samé jedničky. Pro přehlednost označíme řádky, což využijeme u vytváření dalších submatic.

$$G_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix}$$

Řádky submatice G_2 obsahují součiny všech řádků submatice G_1 právě jednou. Protože submatice G_1 má 4 řádky, má submatice G_2 řádků 6 dle vzorce 5.5.

$$G_2 = \binom{m}{I} = \binom{4}{2} = 6 \text{řádků}$$

$$G_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} 1 \cdot 2 \\ 1 \cdot 3 \\ 1 \cdot 4 \\ 2 \cdot 3 \\ 2 \cdot 4 \\ 3 \cdot 4 \end{matrix}$$

Submatice G_3 má 4 řádky a obsahuje součiny vždy 3 řádků submatice G_1 dle vzorce 5.5.

$$G_3 = \binom{m}{I} = \binom{4}{3} = 4 \text{řádky}$$

$$G_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} 1 \cdot 2 \cdot 3 \\ 1 \cdot 2 \cdot 4 \\ 1 \cdot 3 \cdot 4 \\ 2 \cdot 3 \cdot 4 \end{matrix}$$

Výsledná generující matice G vznikne spojením submatic do tvaru

$$G = \begin{bmatrix} G_0 \\ G_1 \\ G_2 \\ G_3 \end{bmatrix}$$

5.2 Oprava kódových slov

Reedovy-Müllerovy kódy se vyznačují jednoduchou a snadno implementovatelnou metodou dekódování. Dekódování je založeno na většinové logice. Pro hledanou hodnotu najdeme soustavu rovnic a rozhodneme se pro 0 nebo 1 tak, aby většina těchto rovnic platila. Reed-Müllerův kód umí opravit 2^{m-r-1}-1 chyb. Předpoklad pro úspěšné dekódování kódového slova je stanovit správný dekódovací algoritmus [3].

Prvním krokem dekódování je určení řádu *r*. Pro kód R-M(*r,m*) se minimální vzdálenost kódu vypočítá dle vzorce 5.7.

$$d = 2^{m-r} \tag{5.7}$$

Přijaté slovo je ve tvaru $w = w_0 w_1 \ldots w_{n-1}$. Délku slova vypočítáme dle vzorce 5.1. Cílem je určit kódové slovo ve tvaru

$$v = \sum_{\substack{i=0 \\ \|i\| \le r}}^{n-1} q_i x_m^{im} \cdots x_2^{i2} x_1^{i1} \tag{5.8}$$

V prvním kroku určíme ty koeficienty q_i, pro které pro které $\|i\| = r$. Jestliže většina z d výrazů dle vzorce 5.7, které získáme dosazením do vzorce 5.8

$$\sum_{j \ni M} w_{j+s} \quad \text{pro } s \in M(n - i - 1) \tag{5.9}$$

je rovna 1 položíme $q_i = 1$, je-li většina rovna 0, 1 položíme $q_i = 0$. V nerozhodném případě položíme $\sum_{j \ni M(i)} w_j$ a ohlásíme, že počet chyb je alespoň *d/2*.

Dalším krokem dekódování je určení koeficientů s indexem váhy *r – 1, r – 2,…, 0*

K určení koeficientů q_i, kde $\|i\| = r - 1$, používáme koeficienty q_j, které jsme určili a položíme

$$w' = w - \sum_{\|i\|=r} q_j x_m^{jm} \cdots x_2^{j2} x_1^{j1} \qquad (5.10)$$

Polynom, který odčítáme, ovšem vyjádříme ve tvaru slova. Nyní aplikujeme předchozí postup na slovo w´. Jestliže většina z $2^{m-(r-1)}$ = 2d výrazů, které získáme dosazením do vzorce (5.11)

$$\sum_{j \ni M(i)} w'_{j+s} \text{ pro } s \in M(n - i - 1) \qquad (5.11)$$

je rovna 1, položíme $q_i = 1$, je-li většina rovna 0 položíme $q_i = 0$. V nerozhodném případě se řídíme rovnicí pro s = 0 a ohlásíme, že počet chyb je alespoň *d/2*.
Tak určíme postupně všechny koeficienty q_i a tím vektor v.

Použití kódu (2,3) dekódujme slovo w = 1111 1010
Každé kódové slovo je tvaru

$$v = q_0 1 + (q_1 x_1 + q_2 x_2 + q_4 x_3) + (q_3 x_1 x_2 + q_5 x_1 x_3 + q_6 x_2 x_3)$$

První krok: určení koeficientů q_3, q_5 *a* q_6 dle vzorce 5.8
Pro q_3 máme M(3) = {0, 1, 2, 3} a M(8 – 3 – 1) = {0, 4}, dle vzorce 5.9, takže
$q_3 = w_0 + w_1 + w_2 + w_3$ (s=0)
$= w_4 + w_5 + w_6 + w_7$ (s=4).

Obě hodnoty jsou 0, a tedy $q_3 = 0$. Podobně určíme $q_5 = 1$ a $q_6 = 0$. A podle [3].platí x_1 = 01010101 a x_3 = 00001111, takže položíme $w' = w - q_5 x_1 x_2$ = 11110000 - 00000101 = 11111111

Druhý krok určení koeficientů q_1, q_2 *a* q_4 dle vzorce 5.10

Pro q_1 máme M(1) = {0, 1} a M(8 – 1 – 1) = {0, 2, 4, 6}, dle vzorce 5.11, takže

$q_3 = w_0 + w_1$ (s=0)

$= w_2 + w_3$ (s=2)

$= w_4 + w_5$ (s=4).

$= w_6 + w_7$ (s=6).

Všechny čtyři výrazy jsou 0 a tedy $q_1 = 0$. Podobně určíme $q_2 = 0$ a $q_4 = 0$.

Položíme

w´´ = w´- o = 11111111

$q_0 = 0$

$v = x_1x_3 + 1 = 1111\ 1010$

V tomto případě w = v

6. Konvoluční kódy

Konvoluce je matematický způsob kombinování dvou signálů za účelem vzniku třetího signálu. Všechny dosud zmíněné kódy jsou kódovány kodéry bez paměti. Jinými slovy výstupní posloupnost je závislá pouze na vstupní informační posloupnosti. Konvoluční kódy jsou samostatnou skupinou kódů, které používají kodéry s pamětí. Výstupní posloupnost konvolučního kódu tedy není závislá pouze na aktuální vstupní informační posloupnosti, ale také na předchozích vstupních informačních posloupnostech.

Konvoluční kód můžeme vytvořit pomocí generující matice, násobením mnohočlenů nebo grafickou metodou pomocí mřížkového diagramu *(trellis diagram).* Nejznámější způsob dekódování je Viterbiho algoritmus.

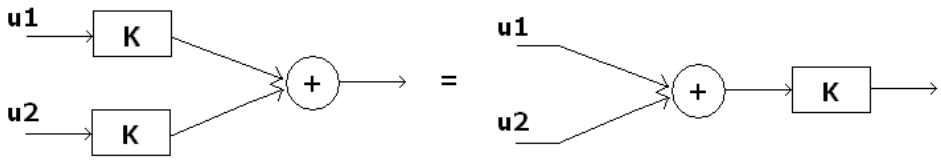
Ještě předtím, než se blíže seznámíme s konvolučními kódy, připomeňme si neobecnější vlastnosti kódu. Touto vlastností je linearita, dle vzorců (6.1 a 6.2).

Pro předpis kódování K musí platit:

1) Kód je lineární tehdy, jsou-li splněny podmínky aditivity a homogenity [4].

Aditivita

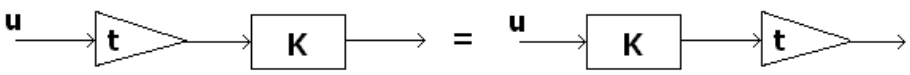
$$K[u1(x) + u2(x)] = K[u1(x)] + K[u2(x)] \tag{6.1}$$



Obrázek 6.1: Aditivita

Homogenita

$$K[t\,u(x)\,] = t\,K[u(x)] \tag{6.2}$$



Obrázek 6.2: Homogenita

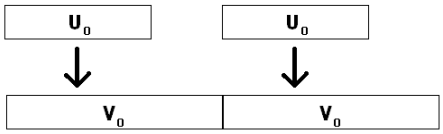
2) Kód je Časově invariantní

Časové zpoždění o *k* kroků na vstupu, vyvolá odpovídající časové zpoždění o *n* kroků na výstupu.

$$K[x^k u(x)\ ] = x^n K[u(x)] \tag{6.3}$$

3) Kód K nemá paměť, jestliže kódové slovo je závislé pouze na aktuální *k*-tici informační posloupnosti a ne na *k*-ticích, které byly na vstupu v době vytvoření předešlých kódových slov.

Jak již bylo zmíněno lineární kódy nemají paměť. Z toho plyne, že stejné informační slovo na více různých pozicích je kódováno pořád stejně, viz obrázek 6.1.



Obrázek 6.1: Blokové schéma informačních posloupností a kódových slov v lineárním kódu.

Konvoluční kódy splňují podmínku linearity, ale mají paměť. Jedná se vlastnost sekvenčního obvodu, kde výstupní hodnota závisí na posloupnosti vstupních hodnot.

Při popisu konvolučních kódů často dochází chybnému výkladu pojmů konvoluční kodér a konvoluční kód. Ty jsou spolu v úzkém vztahu, ale nejde o stejné pojmy. Vysvětleme si tedy pojmy konvoluční kód, konvoluční kodér a operace konvolučního kódování.

Konvoluční kódér - je součástka nebo stroj
Operace konvolučního kódování - je prováděna kodérem
Konvoluční kód - soubor kódových posloupností, které odpovídají všem možným informačním posloupnostem

Předpis pro konvoluční kódování K je vyjádřen linearitou a časovou invariantností. Třetí podmínka o tom, že konvoluční kódy jsou kódovány bez použití kodéru, který neobsahuje paměť, neplatí. Na základě těchto poznatků lze vyslovit definici [5].

Definice: Binárním konvolučním kódem (*n, k*) nazýváme zobrazení K, která binárním polynomům *u(x)* přiřazuje binární polynomy *K[u(x)]* a je

1) lineární

2) časově invariantní $K[x^k u(x)\ ] = x^n K[u(x)]$

Operaci konvolučního kódování provádí kodér, který má následující parametry: n_0, k_0, m

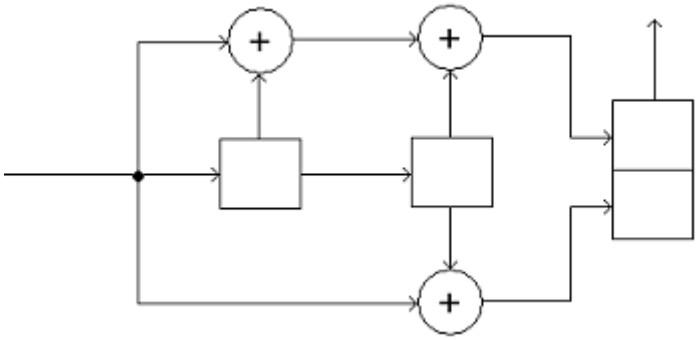
kde

- n_0 – počet výstupů
- k_0 – počet vstupů
- *m* – počet paměťových elementů

Typicky bývá

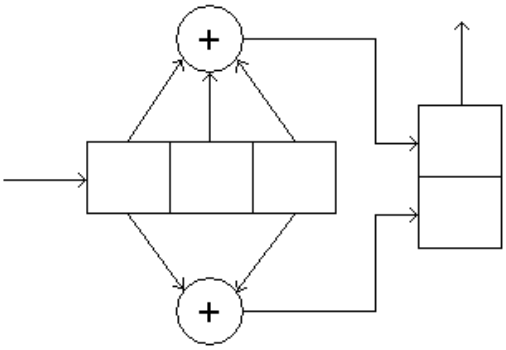
- $k_0 < n_0$
- k_0 a *n*0 bývají menší, než známe hodnoty *n* a *k* z lineárních blokových kódů

Konvoluční kóder má tedy n_0 výstupů, které závisí na $m * k_0$ informačních bitech. Konvoluční kód se mnohdy také zadává rychlostí k_0/n_0 a počtem pamětí *m* (omezující délkou). Konvoluční kóder je sekvenční obvod s k_0 vstupy, n_0 výstupy a m paměťovými elementy.



Obrázek 6.2: Kodér konvolučního kódu (2, 1, 2) s $g(x) = 1 + x + x^2 + x^4 + x^5$

Kodér na obrázku 6.2 se skládá ze dvou klopných obvodů posuvného registru a ze tří obvodů pro sčítání modulo 2 a z dvoubitové výstupní vyrovnávací paměti, která reaguje na jeden informační bit na vstupu dvěma výstupními bity. Mezi vstupním a výstupním slovem je tedy lineární vztah. Časová invariantnost se projevuje tím, že pokud přidáme před vstupní informační posloupnost prodlevu, například jedna nula navíc, na výstupu se to projeví dvěma nulami před kódovým slovem. Na obrázku 6.3 můžeme vidět alternativní zobrazení kodéru.



Obrázek 6.3: Alternativní zobrazení kodéru konvolučního kódu (2,1,2)
s $g(x) = 1 + x + x^2 + x^4 + x^5$

Tento kodér realizuje konvoluční kód (2,1,2) s $g(x) = 1 + x + x^2 + x^4 + x^5$. Generující mnohočlen si můžeme ověřit tím, že přivedeme na vstup logickou jedničku. Na začátku jsou v posuvných registrech nuly. Odezvou na bit hodnoty 1 na vstupu, bude posloupnost bitů 11 na výstupu. Při prvním posuvu bude odezvou posloupnost 10. Pří dalším posuvu pak posloupnost 11. Při třetím posuvu už jsou v kodéru samé nuly. Pomocí mnohočlenů lze tato operace zapsat následovně:

$K[1] = 111011 = g(x) = 1 + x + x^2 + x^4 + x^5$
$K[01] = 00111011$
$K[001] = 0000111011$
Zde vidíme, že časová invariantnost platí.

Linearitu je možné ověřit tak, že vytvoříme rozlad slova 101 na 100 + 001
Platí $K[001] = K[1] + K[001] = 1110001011$

Důležitý parametrem konvolučních kódů je tzv. omezující délka *(constraint length)*, která se značí *m*. Udává kolik *k*-tic zdrojových bitů ovlivňuje jednu *n*-tici kódových bitů na výstupu. Právě touto rychlostí posuvu se realizují různé kodéry, kde v jednom kroku je odlišná reakce na *k* bitů informačního slova a následnému posuvu o *n* bitů kódového slova.

6.1 Vytvoření kódového slova

Při vytváření kódového slova je více možností, jak lze postupovat. Výstupní kódovou posloupnost můžeme vytvořit pomocí mnohočlenů, grafickou metodou je pak tzv. mřížkový diagram *(trellis diagram)* a můžeme využit i generující matice. K vytváření kódového slova využijeme generujícího mnohočlenu, který dostaneme přivedením bitu s hodnotou logické jedničky na vstup kodéru.

Vytvoření kódové slova pomocí mnohočlenů.

Pro výpočet kódových slov je nejvýhodnější vyjádření pomocí mnohočlenů. Informační mnohočlen se násobí s generujícím mnohočlenem. Pokud je splněna vlastnost časové invariantnosti např. $k = 1$

Časová invariantnost v případě $k = 1$ říká, že $K[x^k a(x)] = x^n K[a(x)]$ (6.1)

Z toho plyne, že takový konvoluční kód je úplně popsán svou odpovědí na bit s hodnotou 1
$g_0(x)=K[1]$

Pak platí $K[x]= x^n g_0(x)$
$K[x^2]= x^{2n} g_0(x)$
$K[x^3]= x^{3n} g_0(x)$

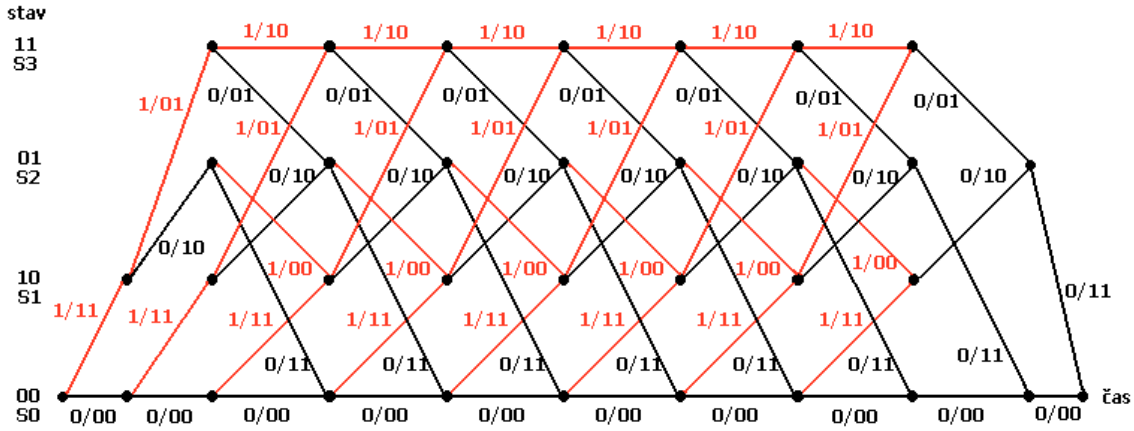
Z linearity tedy dostáváme

$K[a_0 + a_1x + a_2x^2 +\ldots] = (a_0 + a_1x^n + a_2x^{2n} +\ldots) g_0(x)$ (6.2)

Stručněji $K[a(x)] = a(x^n) g0(x)$

Mřížkový diagram *(trellis diagram)*

Názornou grafickou možností popisu je mřížkový diagram. Tento diagram je grafickým znázorněním, které odpovídá úplnému kódovému stromu, ale má sloučené shodné stavy. Mřížkový diagram se skládá z jednotlivých stavů. Tyto stavy jsou propojeny hranami. Horní hrana, která vychází z nějakého stavu, je ohodnocena jedničkou a spodní hrana nulou. Jednotlivé posloupnosti stavů jsou pak nazývány cesta. Na začátku kódování vycházíme z nulového stavu. Tomu odpovídají první čtyři kroky diagramu obrázek 6.4. Stejně jako na začátku vycházíme z nulového stavu, po dokončení kódování se kodér do tohoto stavu musí vrátit, tomu odpovídají poslední 4 kroky. Zdrojová posloupnost bude doplněna o nuly, které se nazývají konec zprávy (*tail of message*).



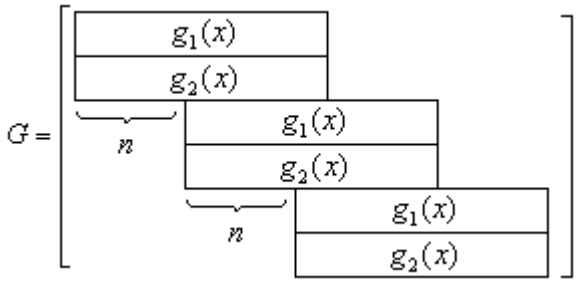
Obrázek 6.4: Znázornění mřížkového diagramu

Pomocí generující matice

Generující matici je tvořena z generujícího mnohočlenu. Pokud je generujících mnohočlenů více, pak jsou psány pod sebe bez posunutí a z nich jsou tvořeny bloky o stejné délce. Další postup při tvoření matice je ovlivněn reakcí počtu výstupních bitů na jeden bit na vstupu v jednom kroku kódování. Pro kodér, který obsahuje dva obvody posuvných registrů neboli má dvě paměti bude tento posuv o dvě pozice doprava. Jak by vypadala generující matice konvolučního kódu (2,1,2) s generujícím mnohočlenem $g(x) = 1 + x + x^2 + x^4 + x^5$, můžeme vidět na obrázku 6.5. Generující matice není ukončena, protože konvoluční kódy nemají blokovou strukturu, to znamená, že kódové slovo nemá pevnou délku.

$$G = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & .. & .. \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & .. & .. \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & .. & .. \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & .. & .. \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .. & .. \\ .. & .. & .. & .. & .. & .. & .. & .. & .. & .. \end{bmatrix}$$

Obrázek 6.5: Ukázka generující matice konvolučního kódu s $g(x) = 1 + x + x^2 + x^4 + x^5$



Obrázek 6.6: Ukázka generující matice konvolučního kódu s generujícími mnohočleny

6.2 Příjem kódového slova

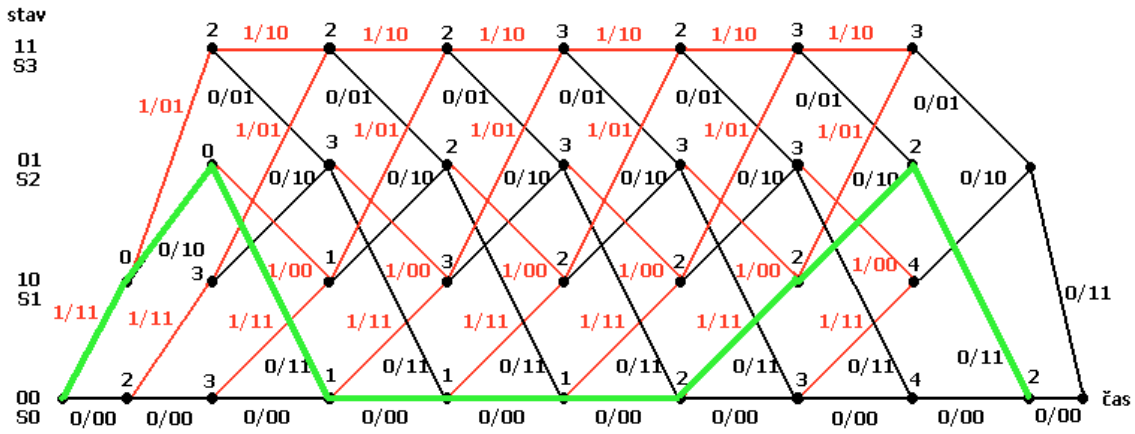
Základní myšlenkou dekódování pro konvoluční kódy je hledat nejpravděpodobnější podobu kódového slova, které bylo odesláno. Jinými slovy přijaté slovo se dekóduje jako to kódové slovo, které je mu nejpodobnější, to znamená, že má od přijatého slova minimální Hammingovu vzdálenost.
Konvoluční kód opravuje *t*-násobné chyby, jestliže dekódování správně určí zdrojovou zprávu, kdykoliv se přijaté slovo liší od vyslaného v nejvýše t bitech [5].

Minimální vzdálenost je rovna nejmenší Hammingově vzdálenosti nenulové kódové poslopnosti. Tato minimální kódová vzdálenost je nazývá volnou kódovou vzdáleností a

označuje se jako d_{free}. Například posloupnost 101111 má minimální kódovou vzdálenost 5 (kódová posloupnost obsahuje pět jedniček). Z toho můžeme říct, že kód opravuje až dvojnásobné chyby v kódové posloupnosti. $d_{free} > 2t$.

Nejznámější způsob dekódování konvolučních kódu je tzv. Viterbiho algoritmus. Je založen na vyhledávání nejmenší Hammingovy vzdálenosti dekódované posloupnosti od přijaté posloupnosti viz obrázek 6.7. Jinými slovy hledáme cestu tímto algoritmem, která bude nejpodobnější kódovému slovu. Využijeme u toho tzv. parcialní vzdálenost, což je vzdálenost slova od začátku a je dána součtem předcházející vzdálenosti a současné vzdálenosti slov odpovídající aktuálnímu vstupu. Pro každý takt vstupní posloupnosti budeme zkoumat Hammingovy vzdálenosti a porovnávat je s hodnotou hrany, které vedou do příslušných stavů. Tyto hodnoty si zapamatujeme a z těch, které mají nejmenší hodnoty vzdálenosti, vytvoříme tzv. aktivní cesty. S těmito aktivními cestami budeme pracovat i v dalších taktech při další vstupní posloupnosti. Hledaná cesta se nazývá přijatá cesta a je zkoumána podle již zmiňovaných kriterií. Porovnáním vzdáleností přijatých cest určíme nejpravděpodobnější cestu s nejmenší vzdáleností a tím i zdrojovou posloupnost pro daný časový úsek. Nesmíme však zapomenout, že již při dalším vstupu se může tato cesta změnit. Proto správnou cestu lze definitivně potvrdit až na konci zprávy.

Nevýhodou Viterbiho algoritmu je jeho náročnost na počet numerických operací. Proto se tohoto algoritmu využívá pro kratší délky kódů.



Obrázek 6.7: Ukázka činnosti Viterbiho algoritmu

7. Animace detekčních a korekčních metod

Animace k detekčním a korekčním metodám jsem vytvářel v programu Adobe Flash. Ten se používá k tvorbě prezentací, animací a interaktivních animací [10]. V současnosti jsou flashové animace velmi oblíbené a to hlavně pro webové aplikace. Jelikož tento software využívá vektorovou grafiku, mají animace malou velikost výsledných souborů.

Flash má také vlastní implementovaný programovací jazyk, který umožňuje dělat robustní a interaktivní animace. Tento jazyk se nazývá ActionScript a je poměrně vyspělý a objektově orientovaný [7].

Flash používá tři formáty:

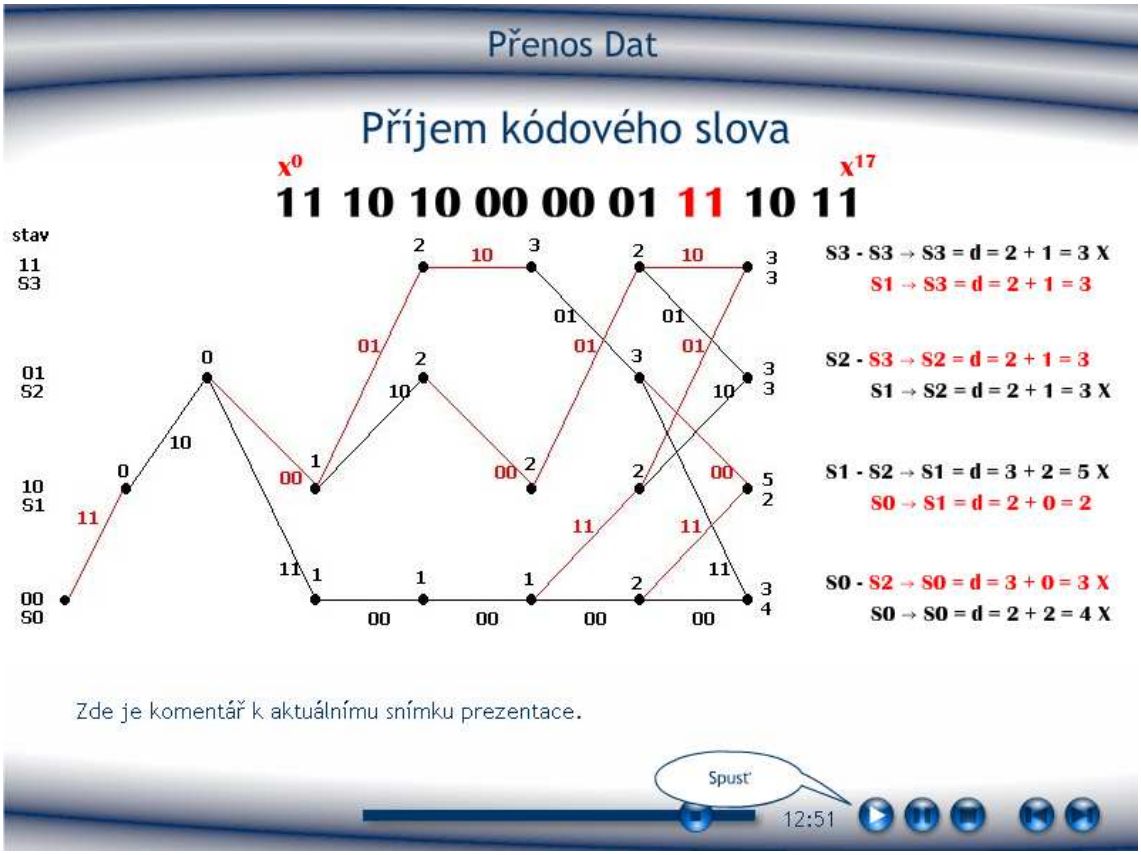
- .fla – v tomto formátu mají výsledné soubory nekomprimovanou velikost. Animace můžeme různě upravovat a pracovat s nimi. Je to v podstatě zdrojový soubor.
- .swf – v tomto formátu už má soubor malou velikost. S animací nelze už nijak pracovat a upravovat, můžeme ji pouze přehrát. K tomu využijeme přehrávač Adobe Flash Player, který je volně k dispozici na internetu.
- .exe – tento formát je určený pro spouštění v operačním systému Windows. Není nutné využívat další přehrávač. Soubor má větší velikost než formát .swf. Formát .exe má přehrávač FlashPlayer implementovaný v sobě.

Animace jsou tvořeny ve verzi Adobe Flash Professional CS4. V současnosti je již novější verze Adobe Flash Professional CS5. V době započetí práce byla aktuální verze CS4 a jelikož není velký rozdíl mezi těmito verzemi, rozhodl jsem se pokračovat ve verzi CS4.

Tyto animace budou sloužit jako doplňkový výukový materiál pro studenty, jak prezenčního, tak i kombinovaného studia. K tomuto účelu jsem vytvořil čtyři animace. První animace, která se vztahuje k CRC kódu obsahuje základní vlastnosti CRC, ukázku vytvoření kódového slova a příklad detekce chyby, která nastala při přenosu. Druhá animace ukazuje princip Hammingových kódů, dále obsahuje základní vlastnosti kódu, ukázku výpočtu zabezpečovacích bitů na základě informačního bloku a následnou detekci a opravu chyby pomocí syndromu. Třetí animaci jsem vytvářil k Reed-Müllerovu kódu, opět jsou v ní zmíněny základní vlastnosti kódu, ukázka vytvoření generující matice a z ní výpočet kódového slova a příklad na opravu vzniklé chyby během přenosu. Poslední animace se zabývá Konvolučními

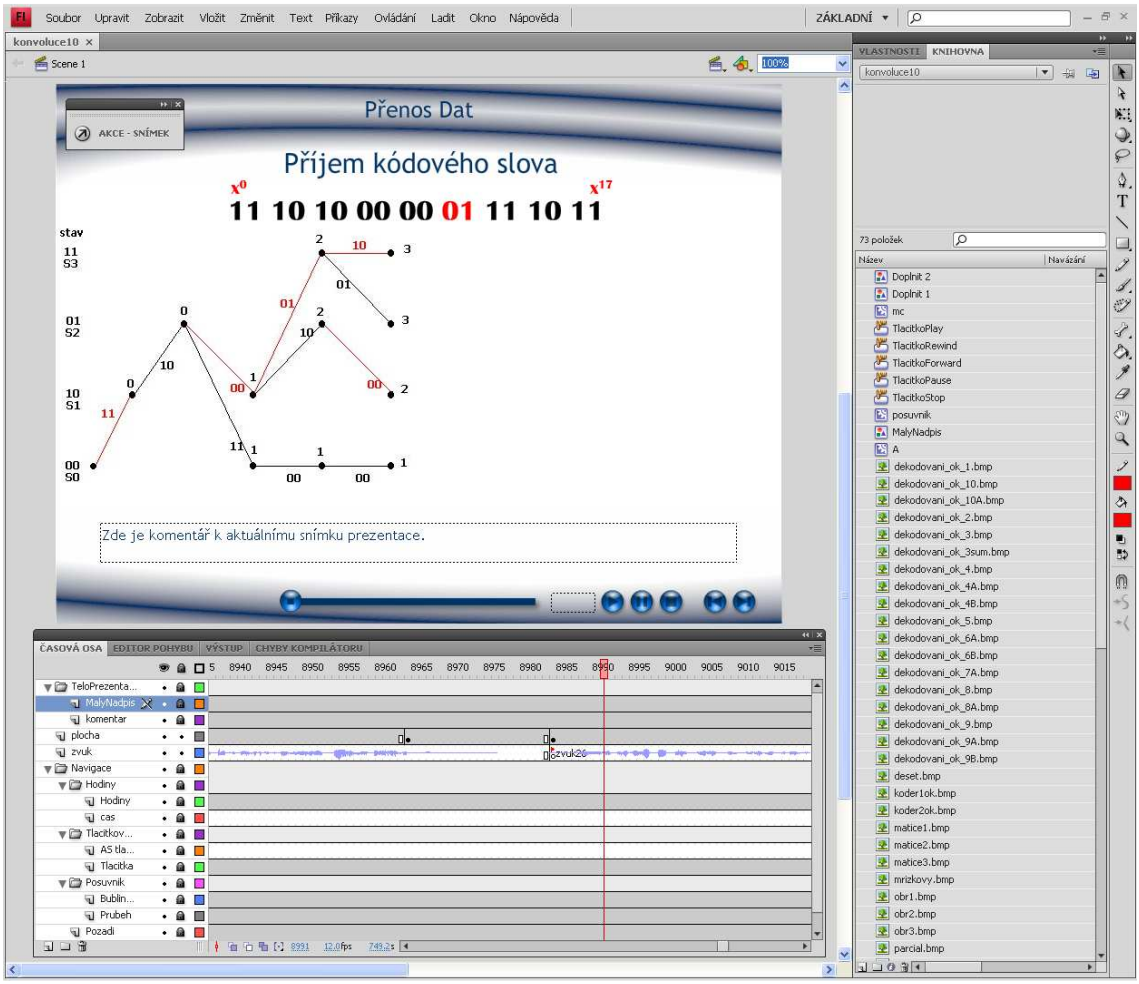
kódy. Obsahuje základní vlastnosti kódu a příklad na vytvoření kódového slova, který je vysvětlen několika způsoby. První způsob je pomocí grafické metody Mřížkového diagramu , Druhý způsob je vytvořením generující matice a posledním způsobem je výpočet pomocí mnohočlenů. Příklad dekódovaní je znázorněn Viterbiho algoritmem. Všechny tyto zmíněné animace obsahují i zvukový komentář.

Při tvoření animací jsem postupoval následovně. Nejdříve jsem si nastudoval princip kódu, jeho základní vlastnosti, způsob vzniku kódového slova, dále způsob dekódování a následnou opravu chyby vzniklé při přenosu. Vytvořil jsem si osnovu, ze které jsem vycházel při tvorbě animací tak, aby obsahovala všechny důležité myšlenky a principy kódu. Z této osnovy jsem vypracoval už podrobnější scénář, který již obsahoval i titulky, které jsem následně doplnil zvukovým komentářem. Poté už pokračovala práce v programu Adobe Flash Professional CS4, kde animace dostala svou finální podobu. Při tvorbě jsem použil jednotnou šablonu, aby animace měly stejný vzhled a barvy. Finální podobu jedné z animací můžeme vidět na obrázku 7.1.



Obrázek 7.1: Ukázka výsledné animace

Pracovní prostředí v programu Adobe Flash CS4 je ukázáno na obrázku 7.2.



Obrázek 7.2: Pracovní prostředí programu Adobe Flash

8. Závěr

Bezchybný přenos dat je důležitou podmínkou pro předání informace na vzdálená místa. K tomuto účelu jsou využívány bezpečnostní kódy, jejichž funkce je v předchozích kapitolách vysvětlena.

Součástí práce je vytvoření čtyř animací pro výukové účely, které vysvětlují funkce a principy konstrukce detekčních a korekčních metod v přenosu dat. V animacích je tento postup vytvoření a dekódování kódového slova vysvětlen na informační posloupnosti, kterou představuje ASCII hodnota písmena A, což je 65 dekadicky. Na této posloupnosti jsem si v praxi mohl ověřit vlastnosti jednotlivých metod.

U Cyklického redundantního součtu jsem ověřil, že se nejedná o blokový kód a neomezuje nás délka vstupní informační posloupnosti. Potvrdilo se, že se jedná o systematický kód, kde je možné pozorovat informační posloupnost, která je zabezpečena cyklickým redundantním součtem a ten je přidán za tuto posloupnost. K výpočtu jsem použil 8-bitové CRC s generujícím polynomem $G(x) = x^8 + x^2 + x + 1$.

Hammingův kód (7,4) je blokový kód, tudíž zvládne zakódovat pro tuto konfiguraci pouze čtyři informační bity a důsledkem toho musela být vstupní informační posloupnost rozdělena na dvě poloviny. Tyto čtyři informační bity jsou zabezpečeny třemi zabezpečovacími bity. Jak jsem si mohl ověřit, kód je systematický, tudíž na prvních čtyřech místech se nachází čtyři informační bity a pak následují tři zabezpečovací.

Při výpočtu kódového slova Reed-Müllerůvým kódem (8,4) jsem si rozdělil vstupní informační posloupnost na dva bloky, protože RM kódy pro tyto parametry umí zakódovat pouze čtyři informační bity, jelikož se jedná o blokový kód a kódové slovo má pevně stanovenou délku. V kódovém slovu nejsou vidět přímo informační bity, protože se jedná o nesystematický kód. Tyto čtyři informační bity jsou zabezpečeny čtyřmi zabezpečovacími bity, což je o jeden zabezpečovací bit více, než tomu bylo u Hammingova kódu.

U konvolučních kódů není důležité, jak dlouhá je vstupní informační posloupnost, protože konvoluční kódy nemají blokovou strukturu. Bity informační posloupnosti nelze z kódového slova odečíst, protože nejsou systematickým kódem. Jelikož jsou tyto kódy kódovány kodéry s pamětí, tedy výstupní kódová posloupnost je závislá i na předchozích stavech, znamená to, že stejná vstupní informační posloupnost nebude zakódována stejným kódovým slovem jako např. u Hammingova nebo Reedova-Müllerova kódu. Při výpočtech jsem využil konvečního kodéru s $g(x) = 1 + x + x^2 + x^4 + x^5$. Kodové slovo lze vytvořit více

způsoby. Prvním způsobem je násobení mnohočlenů, druhý způsob je pomocí generující matice a poslední způsob je pomocí grafické metody mřížkového diagramu. U dekódování jsem využil Vitterbiho algoritmu, který pracuje na principu parciálních vzdáleností.

V příloze jsem uvedl příklady k jednotlivým metodám, na kterých jsem demonstroval vznik kódového slova a způsob opravy chyby.

Jelikož práce má sloužit k výukovým účelům, bylo mojí snahou, aby byla srozumitelná, věcná a vedla k pochopení problematiky bezpečnostních kódů.

Literatura

[1] VLČEK, Karel. *Komprese a kódová zabezpečení v multimediálních komunikacích*. Praha : BEN, 2000. 226 s.

[2] Cyclic redundancy check. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, , last modified on 28. 5. 2011 [cit. 2011-05-05]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Cyclic_redundancy_check>

[3] ADÁMEK, Jiří. *Kódování*. Praha : Nakladatelství technické literatury, 1989. 190 s.

[4] VLČEK, Karel. *Teorie informace a kódování*. Ostrava : VŠB - Technická Univerzita Ostrava, 1998. 128 s.

[5] ADÁMEK, Jiří. *Kódování a teorie informace* . Praha : Vydavatelství ČVUT, 1994. 209 s.

[6] Error-correcting codes In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 25 April 2010, [cit. 2010-04-25]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Error-correcting_codes>.

[7] Používání programu Flash CS4 Professional. Adobe, 2009 [cit. 2010-04-25]. Dostupné z WWW: <http://help.adobe.com/cs_CZ/Flash/10.0_UsingFlash/WS0CC7D962-6484-46f0-9596-BB27B21C4926a.html>.

[9] PLUHÁČEK, Alois. *Aritmetika a Kódy*. Praha : Vydavatelství ČVUT, 1981. 200 s.

[10] VELECKÝ, Jakub. *Poznejme flash*. Svitavy : Grantis, 2009. 145 s.

Přílohy

Seznam příloh

Příloha 1 - Hammingův kód (7,4)

V této části si ukážeme, jak zakódujeme písmenko A, kterému odpovídá ASCII hodnota 65, kterou si vyjádříme binárně 0100 0001. Tuto posloupnost zakódujeme pomocí Hammingova Kódu (7,4). Následovat bude příklad na dekódování přijatého slova. Jelikož Hammingův kód (7,4) dokáže přenést pouze 4 informační bity, je třeba tuto informaci kódovat dvakrát. Nejdříve přeneseme horní 4 bity a potom zbývající čtveřici bitů. K těmto 4 informačním bitům dopočítáme 3 zabezpečovací bity.

Vytvoření kódového slova

Informační posloupnost tvořená 4 bity ($b_7\ b_6\ b_5\ b_4$) u(x) = 0100, k této posloupnosti dopočítáme zabezpečovací bity dle (A.1)

$$[v_5 v_6 v_7] = [v_1 v_2 v_3 v_4] \cdot \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{A.1}$$

Rovnice k vypočítání zabezpečovacích bitů v_5 v_6 a v_7

$$v_5 = v_2 \oplus v_3 \oplus v_4 \tag{A.2}$$

$$v_6 = v_1 \oplus v_3 \oplus v_4 \tag{A.3}$$

$$v_7 = v_1 \oplus v_2 \oplus v_4 \tag{A.4}$$

Za v_1 až v_4 dosadíme informační bity b_7 až b_4 dle vzorce A.1

$$[v_5 v_6 v_7] = \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \\ 0 & 1 & 0 & 0 \\ b_7 & b_6 & b_5 & b_4 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Dosadíme bity b_6 až b_4 potřebné pro výpočet zabezpečovacího bitu v_5 dle vzorce A.2

$$v_5 = v_2 \oplus v_3 \oplus v_4 = \begin{bmatrix} v_2 & v_3 & v_4 \\ 1 & 0 & 0 \\ b_6 & b_5 & b_4 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

a provedeme výpočet vektoru a matice, násobíme první prvek vektoru s prvním sloupcem matice, následuje druhý prvek vektoru s druhým sloupcem matice a nakonec třetí prvek vektoru se třetím sloupcem matice.

$$v_5 = \begin{matrix} v_2 \\ 1 \\ b_6 \end{matrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \oplus \begin{matrix} v_3 \\ 0 \\ b_5 \end{matrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \oplus \begin{matrix} v_4 \\ 0 \\ b_4 \end{matrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

Zde můžete vidět celý výpočet.

$$v_5 = \underbrace{1 \cdot 0 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1}_{0+1+1+1=0} \oplus \underbrace{0 \cdot 1 + 0 \cdot 0 + 0 \cdot 1 + 0 \cdot 1}_{0+0+0+0=0} \oplus \underbrace{0 \cdot 1 + 0 \cdot 1 + 0 \cdot 0 + 0 \cdot 1}_{0+0+0+0=0}$$

K tomu abychom dostali hledaný zabezpečovací bit v_5 nám už nám stačí jediné a to sečíst výsledky těchto skupin 1+ 0 + 0 = 1 a máme výslednou hodnotu bitu v_5.

Hodnota V_5 je tedy 1 + 0 + 0 = 1

Stejným způsobem vypočítáme zabezpečovací bit v_6 k výpočtu použijeme informační bity v_1,v_3 a v_4 dle vzorce A.3. Za ně dosadíme bity b_7 b_5 a b_4 potřebné k výpočtu

$$v_6 = v_1 \oplus v_3 \oplus v_4 = \begin{bmatrix} v_1 & v_3 & v_4 \\ 0 & 0 & 0 \\ b_7 & b_5 & b_4 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

a nyní provedeme výpočet vektoru a matice, kde násobíme prvky vektoru se sloupci matice.

$$v_6 = \begin{matrix} v_1 \\ 0 \\ b_7 \end{matrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \oplus \begin{matrix} v_3 \\ 0 \\ b_5 \end{matrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \oplus \begin{matrix} v_4 \\ 1 \\ b_4 \end{matrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

Zde vidíme rozepsanou celou operaci

$$v_6 = \underbrace{0\cdot 0+0\cdot 1+0\cdot 1+0\cdot 1}_{0+0+0+0=0} \oplus \underbrace{0\cdot 1+0\cdot 0+0\cdot 1+0\cdot 1}_{0+0+0+0=0} \oplus \underbrace{0\cdot 1+0\cdot 1+0\cdot 0+0\cdot 1}_{0+0+0+0=0}$$

Abychom dostali hledaný zabezpečovací bit v_6 už nám stačí jediné a to sečíst výsledky těchto skupin 0 + 0 + 0 = 0 a dostáváme výslednou hodnotu bitu v_6.

Hodnota V_6 je tedy 0 + 0 + 0 = 0

Stejně postupujeme i u zabezpečovacího bitu v_7 k výpočtu použijeme informační bity v_1 v_2 v_4 dle vzorce A4

Dosadíme za ně bity b7 b6 a b4

$$v_7 = v_1 \oplus v_2 \oplus v_4 = \begin{bmatrix} v_1 & v_2 & v_4 \\ 0 & 1 & 0 \\ b_7 & b_6 & b_4 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

a provedeme výpočet vektoru a matice

$$v_7 = \begin{matrix} v_1 \\ 0 \\ b_7 \end{matrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \oplus \begin{matrix} v_2 \\ 1 \\ b_6 \end{matrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \oplus \begin{matrix} v_4 \\ 0 \\ b_4 \end{matrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

Zde vidíme rozepsanou celou operaci

$$v_7 = \underbrace{0\cdot 0+0\cdot 1+0\cdot 1+0\cdot 1}_{0+0+0+0=0} \oplus \underbrace{1\cdot 1+1\cdot 0+1\cdot 1+1\cdot 1}_{1+0+1+1=1} \oplus \underbrace{0\cdot 1+0\cdot 1+0\cdot 0+0\cdot 1}_{0+0+0+0=0}$$

Hodnota V_7 je tedy 0 + 1 + 0 = 1

Horní část odeslaného kódového slova má tedy podobu 0100 101

Tento postup se bude opakovat i pro zbylou čtveřici bitů b3 až b0, kde dosadíme informačni posloupnost u = 0001 = (b_3 b_2 b_1 b_0) dle vzorce A.1

$$[v_5 v_6 v_7] = \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \\ 0 & 0 & 0 & 1 \\ b_3 & b_2 & b_1 & b_0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

pro výpočet zabezpečovacího bitu v_5 dosadíme dle vzorce A.2 bity b_2 až b_0

$$v_5 = v_2 \oplus v_3 \oplus v_4 = \begin{bmatrix} v_2 & v_3 & v_4 \\ 0 & 0 & 1 \\ b_2 & b_1 & b_0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

a provedeme výpočet vektoru a matice

$$v_5 = \begin{matrix} v_2 \\ 0 \\ b_2 \end{matrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \oplus \begin{matrix} v_3 \\ 0 \\ b_1 \end{matrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \oplus \begin{matrix} v_4 \\ 1 \\ b_0 \end{matrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

Zde vidíme rozepsanou celou operaci

$$v_5 = \underbrace{0 \cdot 0 + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 1}_{0+0+0+0=0} \oplus \underbrace{0 \cdot 1 + 0 \cdot 0 + 0 \cdot 1 + 0 \cdot 1}_{0+0+0+0=0} \oplus \underbrace{1 \cdot 1 + 1 \cdot 1 + 1 \cdot 0 + 1 \cdot 1}_{1+1+0+1=1}$$

Hodnota v_5 je tedy 0 + 0 + 1 = 1

Pro výpočet zabezpečovacího bitu v_6 dosadíme bity b_3 b_1 a b_0 dle vzorce A.3

$$v_6 = v_1 \oplus v_3 \oplus v_4 = \begin{bmatrix} v_1 & v_3 & v_4 \\ 0 & 0 & 1 \\ b_3 & b_1 & b_0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

dále provedeme výpočet vektoru a matice

$$v_6 = \begin{matrix} v_1 \\ 0 \\ b_3 \end{matrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \oplus \begin{matrix} v_3 \\ 0 \\ b_1 \end{matrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \oplus \begin{matrix} v_4 \\ 1 \\ b_0 \end{matrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

Zde vidíme rozepsanou celou operaci

$$v_6 = \underbrace{0 \cdot 0 + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 1}_{0+0+0+0=0} \oplus \underbrace{0 \cdot 1 + 0 \cdot 0 + 0 \cdot 1 + 0 \cdot 1}_{0+0+0+0=0} \oplus \underbrace{1 \cdot 1 + 1 \cdot 1 + 1 \cdot 0 + 1 \cdot 1}_{1+1+0+1=1}$$

Hodnota V_6 je tedy 0 + 0 + 1 = 1

Pro výpočet zabezpečovacího bitu v_7 a dosadíme bity b_3 b_2 a b_0 dle vzorce A.4

$$v_7 = v_1 \oplus v_2 \oplus v_4 = \begin{bmatrix} v_1 & v_2 & v_4 \\ 0 & 0 & 1 \\ b_3 & b_2 & b_0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

a provedeme výpočet vektoru a matice

$$v_7 = \begin{matrix} v_1 \\ 0 \\ b_3 \end{matrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \oplus \begin{matrix} v_2 \\ 0 \\ b_2 \end{matrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \oplus \begin{matrix} v_4 \\ 0 \\ b_0 \end{matrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

Zde vidíme rozepsanou celou operaci

$$v_7 = \underbrace{0 \cdot 0 + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 1}_{0+0+0+0=0} \oplus \underbrace{0 \cdot 1 + 0 \cdot 0 + 1 \cdot 0 + 0 \cdot 1}_{0+0+0+0=0} \oplus \underbrace{1 \cdot 1 + 1 \cdot 1 + 1 \cdot 0 + 1 \cdot 1}_{1+1+0+1=1}$$

Hodnota v_7 je tedy 0 + 0 + 1 = 1

Zbývající čtveřice má tedy podobu 0001 111

A zde už můžeme vidět celou posloupnost, jak bude zakódováno písmenko A pomocí Hammingova kódu (7,4)

Celá posloupnost pak tedy bude 0100 101 0001 111

Příjem kódového slova

Předpokládejme, že při přenosu došlo k jednonásobné chybě a přijali jsme slovo w = [0110 101], Vynásobením kontrolní matice H a přijatého slova w, které převedeme na transponovaný tvar, dostaneme hodnotu syndromu, viz rovnice A.5. Tento syndrom musí být roven sloupci kontrolní matice, jehož pořadí v matici je rovné pořadí chybného bitu v přijatém slově. .

$$s^T = H\, w^T \tag{A.5}$$

$$s^T = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \\ w_7 \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix}$$

Dosadíme dle rovnice A.5

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

Jak můžeme vidět k chybě došlo na třetím místě. Na této pozici provedeme opravu za správný znak a tím opravíme přijaté slovo na správné kódové slovo [0100 101].

Příloha 2 Příklad - CRC

V této části si ukážeme, jak zakódujeme písmeno A, kterému odpovídá ASCII hodnota 65, kterou si vyjádříme binárně 0100 0001. Tuto posloupnost zakódujeme pomocí CRC-8. Následovat bude příklad na dekódování přijatého slova.

Vytváření kódového slova

Základní vztahy pro vyslání kódového slova

$$B(x) = x^r M(x) \oplus R(x) \tag{B.1}$$

$$\frac{B(x)}{G(x)} = Q(x) \tag{B.2}$$

$$\frac{x^r M(x)}{G(x)} = Q(x) + \frac{R(x)}{G(x)} \tag{B.3}$$

Chceme určit vyslané kódové slovo B(x). Jelikož chceme spočítat CRC pro 8 bitovou zprávu obsahující písmeno A jehož ASCII kód je 65 dekadicky nebo 0100 0001 binárně, použijeme
8-bitové CRC.

Předpokládejme 8-bitové CRC s generujícím polynomem $G(x) = x^8 + x^2 + x + 1$
Řád polynomu je tedy 8

Zde můžeme vidět jak vypadá informační polynom
$(65)_D = (0100\ 0001)_B \rightarrow M(x) = 0 \cdot x^7 + 1 \cdot x^6 + 0 \cdot x^5 + 0 \cdot x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0 = x^6 + 1$

Před vlastním výpočtem je třeba doplnit blok informačních symbolů $M(x)$ zprava osmi nulovými bity. Nebo-li vynásobit informační blok řádem polynomu dle vzorce B3.

$$\frac{x^r M(x)}{G(x)} = \frac{x^8\left(x^{14}+x^8\right)}{x^8+x^2+x+1}$$

A nyní přejdeme k výpočtu, kde dělíme polynom polynomem následujícím způsobem:

$$\left(x^{14}+x^8+x^7+x^6\right) \div \left(x^8+x^2+x+1\right) = x^6$$
$$\oplus\ x^{14}+x^8+x^7+x^6$$
$$x^7+x^6 \rightarrow R(x) = x^7+x^6$$

Abychom získali vyslané kódové slovo B(x) musíme tento zbytek po dělení R(x) přičíst k Bloku informačních symbolů vynásobených řádem polynomu dle vzorce B.1. Zde už vidíme kompletní kódové slovo B(x)

$$B(x) = x^r M(x) \oplus R(x) = x^8\left(x^{14}+x^8\right) \oplus x^7+x^6 = x^{14}+x^8+x^7+x^6 = B(x)$$

Příjem kódového slova

Zde můžeme vidět základní vztahy pro příjem kódového slova

$$B'(x) = B(x) \oplus E(x)$$

(B.4)

$$\frac{B'(x)}{G(x)} = Q'(x) + \frac{S(x)}{G(x)}$$

(B.5)

Pomocí těchto vztahů vypočítáme kontrolní syndrom. Pokud je

S(x)=0 → při přenosu nedošlo k chybě nebo došlo k nedetekovatelné chybě

S(x)≠ 0 → při přenosu došlo k chybě

Předpokládejme, že v našem předchozím odesílaném slovu nastala chyba a přijali jsme slovo

$$B'(x) = x^8 + x^7 + x^6$$

Spočítáme-li kontrolní syndrom pro toto přijaté slovo a na základě hodnoty syndromu zjistíme, zda při přenosu došlo k chybě. Je zadán vytvářecí polynom $G(x) = x^8 + x^2 + x + 1$

Resp. Vyberte přijaté kódové slovo B´(x) na základě znalosti vyslaného kódového slova $B(x) = x^{14} + x^8 + x^7 + x^6$ a chybového syndromu $E(x) = x^{14}$. Pro toto slovo určete kontrolní syndrom S(x). Dle vzorce B.4 zavedu uměle chybu.

$$B'(x) = B(x) \oplus E(x) = x^{14} + x^8 + x^7 + x^6 \oplus x^{14} = x^8 + x^7 + x^6$$

Opět provedu dělení dvou polynomů, abych zjistil hodnotu kontrolního syndromu S(x) dle vzorce B.5

$$\frac{B'(x)}{G(x)} = \frac{x^8 + x^7 + x^6}{x^8 + x^2 + x + 1} =$$

$$= (x^8 + x^7 + x^6) \div (x^8 + x^2 + x + 1) = 1$$
$$\oplus (x^8 + x^2 + x + 1)$$
$$x^7 + x^6 + x^2 + x + 1 \rightarrow S(x) \neq 0 \rightarrow Chyba$$

CRC se používá pouze k detekci chyb během přenosu či ukládání dat, nikoliv však k opravě těchto chyb.

Příloha 3 Příklad - Reed-Müllerův kód (8,4)

V této části si ukážeme, jak zakódujeme písmeno A, kterému odpovídá ASCII hodnota 65, kterou si vyjádříme binárně 0100 0001. Tuto posloupnost zakódujeme pomocí Reed-Müllerova kódu (8,4)Následovat bude příklad na dekódování přijatého slova.

Jak bylo zmíněno v úvodu, využijeme k tomu Reed-Müllerův kód (8,4)
Tento kód má tyto parametry
Reed-Müllerův kód (8,4)

m = 3
$n = 2^m = 2^3 = 8$
r = 1
$k = 1 + \binom{3}{1} = 1 + 3 = 4$

Vytvoření kódového slova

G0 je tedy vektor jedniček o délce 8 bitů

$$G_0 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

G1je submatice o rozměru m x 2^m kde sloupec vlevo obsahuje samé nuly a sloupec vpravo samé jedničky

$$G_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Výsledná matice G vznikne spojením těchto dvou submatic

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Jelikož Reed-Müllerův kód (8,4) dokáže přenést pouze 4 informační bity, budeme tuto informaci kódovat na 2x. Nejprve přenesme horní 4 bity a potom následující čtveřici bitů.
Těmto 4 informačním bitům dopočítáme zabezpečovací bity. Kódové slovo pak určíme operací, kdy vynásobíme informační blok a generující matici. Důležité je si správně označit všechny bity. Před násobením nesmíme zapomenout tento informační blok zrcadlově otočit!

Celá operace násobení pak bude vypadat takto

$$B(x) = M(x) \cdot G = \begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\ 0 & 0 & 1 & 0 \\ b_4 & b_5 & b_6 & b_7 \end{bmatrix} \cdot \begin{bmatrix} s0 & s1 & s2 & s3 & s4 & s5 & s6 & s7 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} =$$

$= 0 \cdot 1 + 0 \cdot 0 + 1 \cdot 0 + 0 \cdot 0 = 1 = b0$

$0 \cdot 1 + 0 \cdot 0 + 1 \cdot 0 + 0 \cdot 1 = 1 = b1$

$0 \cdot 1 + 0 \cdot 0 + 1 \cdot 1 + 0 \cdot 0 = 1 = b2$

$0 \cdot 1 + 0 \cdot 0 + 1 \cdot 1 + 0 \cdot 1 = 1 = b3$

$0 \cdot 1 + 0 \cdot 1 + 1 \cdot 0 + 0 \cdot 0 = 1 = b4$

$0 \cdot 1 + 0 \cdot 1 + 1 \cdot 0 + 0 \cdot 1 = 1 = b5$

$0 \cdot 1 + 0 \cdot 1 + 1 \cdot 1 + 0 \cdot 0 = 1 = b6$

$0 \cdot 1 + 0 \cdot 1 + 1 \cdot 1 + 0 \cdot 1 = 1 = b7$

Za bity x_0 až x_3 dosadíme bity b_7 až b_4, které nesmíme zapomenout dosadit v opačném pořadí. Následuje násobení informačního bloku s generující maticí, kde násobíme každý prvek informačního bloku s jednotlivými sloupci generující matice. Po vynásobení každého sloupce a následného sečtení jednotlivých bitů dostáváme příslušný bit kódového slova. Tento bit odpovídá označení sloupce generující matice.

Celé kódové slovo je tedy 1100 1100

Nyní provedeme totéž se zbylou čtveřici bitů. Za bity x_0 až x_3 dosadíme zbylé 4 bity B_3 až B_0, které opět nesmíme zapomenout dosadit v opačném pořadí.

$$B(x) = M(x) \cdot G = \begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\ 1 & 0 & 0 & 0 \\ b_0 & b_1 & b_2 & b_3 \end{bmatrix} \cdot \begin{bmatrix} s0 & s1 & s2 & s3 & s4 & s5 & s6 & s7 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} =$$

$= 1 \cdot 1 + 0 \cdot 0 + 0 \cdot 0 + 0 \cdot 0 = 1 = b0$

$1 \cdot 1 + 0 \cdot 0 + 0 \cdot 0 + 0 \cdot 1 = 1 = b1$

$1 \cdot 1 + 0 \cdot 0 + 0 \cdot 1 + 0 \cdot 0 = 1 = b2$

$1 \cdot 1 + 0 \cdot 0 + 0 \cdot 1 + 0 \cdot 1 = 1 = b3$

$1 \cdot 1 + 0 \cdot 1 + 0 \cdot 0 + 0 \cdot 0 = 1 = b4$

$1 \cdot 1 + 0 \cdot 1 + 0 \cdot 0 + 0 \cdot 1 = 1 = b5$

$1 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 0 = 1 = b6$

$1 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 = 1 = b7$

Celé kódové slovo je tedy 1111 1111

Zde můžeme vidět celou posloupnost, jak bude zakódováno písmeno A pomocí Reed-Müllerova kódu (8,4). V kódovém slovu nevidíme přímo informační blok, jelikož se jedná o nesystematický kód.

Celé kódová posloupnost má tvar 1100 1100 1111 1111

Příjem kódového slova

Reed-Müllerův kód umí opravit 2^{m-r-1}-1 chyb

Dekódování je založeno na většinové logice. Pro hledanou hodnotu najdeme soustavu rovnic a rozhodneme se pro 0 nebo 1 tak, aby většina těchto rovnic platila.

Ukažme si to na našem případu pro m = 3 a r = 1. Ze vzorce můžeme vidět, že pro tyto parametry kód umí opravit jednu chybu.
$2^{3-1-1}-1 =$ 1 chyba

Předpokládejme tedy, že při přenosu došlo k jednonásobné chybě a přijali jsme slovo 1110 1100 ve tvaru (b0 – b7)

Při dekódování určíme majoritní hodnotu z rovnic vytvořených pro každý informační bit. Rovnice jsou určeny ze sloupců (resp. vektorů) s Hammingovou vzdáleností $d_{min} = 1$, které se liší v právě požadované hodnotě.

$x_{3,1} = w_0 + w_1 = 0 + 0 = 0$
$x_{3,2} = w_2 + w_3 = 1 + 1 = 0$
$x_{3,3} = w_4 + w_5 = 0 + 1 = 1$
$x_{3,3} = w_6 + w_7 = 1 + 1 = 0$

$x_{2,1} = w_0 + w_2 = 0 + 1 = 1$
$x_{2,2} = w_1 + w_3 = 0 + 1 = 1$
$x_{2,3} = w_4 + w_6 = 0 + 1 = 1$
$x_{2,3} = w_5 + w_7 = 1 + 1 = 0$

$x_{1,1} = w_0 + w_4 = 0 + 0 = 0$
$x_{1,2} = w_1 + w_5 = 0 + 1 = 1$
$x_{1,3} = w_2 + w_6 = 1 + 1 = 0$
$x_{1,3} = w_3 + w_7 = 1 + 1 = 0$

Do těchto rovnic dosadíme bity přijatého slova a určíme majoritní hodnotu.
$x_3 = 0$
$x_2 = 1$
$x_1 = 0$

Určení majoritní hodnoty pro x1, x2 a x3 je předpokladem pro následné určení x0. To je dané sčítáním hodnoty w0 až w7 a příslušných majoritních hodnot x1, x2 a x3 reprezentovaných v daném sloupci submatice matice G1. Nyní za tyto rovnice dosadíme a vypočítáme poslední informační bit x0.

$x_{0,1} = w_0 = 0$
$x_{0,2} = w_1 + x_3 = 0 + 0 = 0$
$x_{0,3} = w_2 + x_2 = 1 + 1 = 0$
$x_{0,4} = w_3 + x_2 + x_3 = 1 + 0 + 1 = 0$
$x_{0,5} = w_4 + x_1 = 0 + 0 = 0$

$x_{0,6} = w_5 + x_1 + x_3 = 1 + 0 + 0 = 1$

$x_{0,7} = w_6 + x_1 + x_2 = 1 + 0 + 1 = 0$

$x_{0,8} = w_7 + x_1 + x_2 + x_3 = 1 + 0 + 1 + 0 = 0$

$x_0 = 0$

Jak můžeme vidět z výsledků majoritní hodnota těchto rovnic je 0, tudíž poslední informační bit x0 je také nula.

Kdyby bylo přijaté slovo 1110 1110, znamená to, že při přenosu došlo ke dvojnásobné chybě. Kdybychom ovšem nyní chtěli najít blok informačních bitů, z výpočtu bitu x3 je patrné, že nemůžeme určit majoritní prvek. Ve výpočtu tedy nemá smysl dál pokračovat. RM kód pro parametry m = 3 a r = 1 není schopen opravovat dvojnásobné a vícenásobné chyby.

Příloha 4 Příklad - Konvoluční kód

V této části si ukážeme, jak zakódujeme písmeno A, kterému odpovídá ASCII hodnota 65, kterou si vyjádříme binárně 0100 0001. Tuto posloupnost zakódujeme pomocí konvolučního kódu. Následovat bude příklad na dekódování přijatého slova. K vytvoření kódového slova využijeme kodéru Konvolučního kódu (2,1,2) s generujícím mnohočlenem
$g(x) = 1 + x + x^2 + x^4 + x^5$

Vytvoření kódového slova

Jendou z možností, jak tuto posloupnost vytvořit je pomocí **násobení mnohočlenů**. Informační mnohočlen budeme násobit s generujícím mnohočlenem. Musíme dodržet vlastnost časové invariantnosti pro náš příklad se k = 1

Časová invariantnost v případě k = 1 říká, že $K[x^k a(x)] = x^n K[a(x)]$

Z toho plyne, že takový konvoluční kód je úplně popsán svou odpovědí na jednu jedničku
$g_0(x) = K[1]$

Pak platí $K[x]= x^n g_0(x)$
$K[x^2]= x^{2n} g_0(x)$
$K[x^3]= x^{3n} g_0(x)$

Z linearity tedy dostáváme

$K[a_0 + a_1x + a_2x^2 + \ldots] = (a_0 + a_1x^n + a_2x^{2n} + \ldots) g_0(x)$

Stručněji $K[a(x)] = a(x^n) g0(x)$

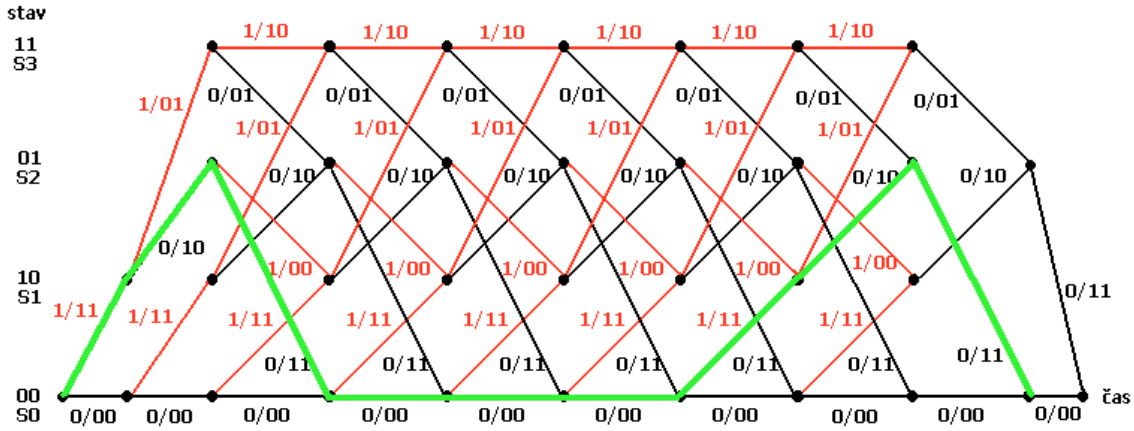
Stejným způsobem si přepočítáme informační mnohočlen pro dané parametry n = 2 a k = 1
A nyní můžeme začít počítat kódové slovo. Při násobení se koeficienty sčítají.

$K[1+x^6] = (1+x^{12})\, g0(x)$

$(1+x^{12})\,(1+x+x^2+x^4+x^5) = 1+x+x^2+x^4+x^5+x^{12}+x^{13}+x^{14}+x^{16}+x^{17}$

Převedeno na bitovou posloupnost: 11 10 11 00 00 00 11 10 11

Další možností je tzv. **Mřížkový diagram**. Tento diagram je grafickým znázorněním, které je ekvivalentní úplnému kódovému stromu, ale má sloučené shodné stavy. Kodér je na počátku v nulovém stavu. Zdrojová posloupnost bude doplněna o nuly, které se nazývají konec zprávy, protože kódová posloupnost pokračuje tak, aby se kodér dostal do nuly. Pro přehlednost červeně jsou ohodnocené hrany hodnotou 1 a černě hodnotou 0.



Obrázek D.1: Výsledná cesta Mřížkovým diagramem

Výstupní kódová posloupnost tedy je 11 10 11 00 00 00 11 10 11, viz obrázek D.1

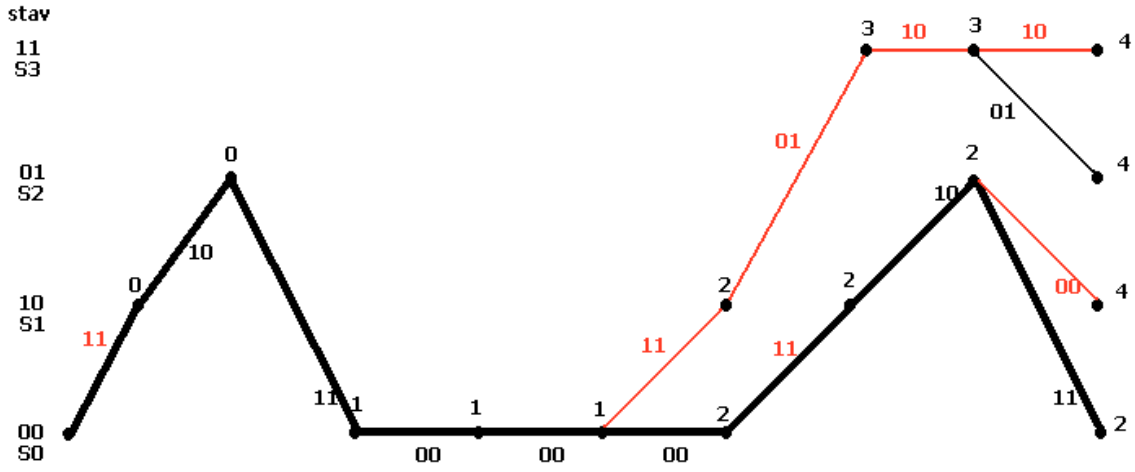
Další možností je vytvořit **generující matici**. Tu sestavíme z generujícího mnohočlenu. V případě, že je generujících mnohočlenů více, jsou napsány pod sebe bez posunutí a tvoří blok o stejné délce. Další pokračování generující matice je tvořeno s ohledem na počet výstupních bitů kódového slova v jednom kroku kódování. V našem případě to tedy bude o dvě pozice doprava. Generující matice není ukončena, protože konvoluční kódy nemají blokovou strukturu. Výpočet výstupní kódové posloupnosti provedeme na základě vynásobení informačního vektoru se všemi sloupci generující matice.

$$\begin{bmatrix} x^0 & x^1 & x^2 & x^3 & x^4 & x^5 & x^6 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Příjem kódového slova

Nejznámější způsob dekódování konvolučních kódů je Viterbiho algoritmus.Viterbi dekodér vybírá takovou cestu přes mřížkový diagram, která má minimální vzdálenost mezi přijatým slovem a kódovým slovem. Je založen na parcialní vzdálenost – je to vzdálenost slova od začátku a je dána součtem předcházející vzdálenosti a současné vzdálenosti slov odpovídající aktuálnímu vstupu.

Nechť obsahuje 2 chybné bity a je ve tvaru: 11 10 10 00 00 01 11 10 11



Obrázek D.2: Znázornění činnosti Viterbiho algoritmu

Po přijetí zprávy je kodér v nulovém stavu. Výsledná minimální parciální vzdálenost je 2. Výsledná opravená posloupnost je tedy posloupnost je tedy 11 10 11 00 00 00 11 10 11, viz obrázek D.2. Jak můžeme vidět, byly opraveny obě chyby. Pokud by jsme si chtěli dekódovat odeslané informační slovo, půjdeme stejnou cestou. Červené hrany mají hodnotu jedna a černé nula. Odeslaná posloupnost je tedy 10 00 00 100, což odpovídá $1 + x^6$ a to se rovná našemu písmenu A.